Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

Лабораторная работа №1 по курсу «Компьютерная графика»

Тема: Построение изображений 2D-кривых.

Студент: Тимофеев А. В.

Преподаватель: Филиппов Г. С.

Группа: М80-307Б

Дата:

Оценка:

Подпись:

Москва, 2022

**Постановка задачи**

Написать и отладить программу, строящую изображение заданной замечательной кривой.  
Вариант 3: x = a\* cos3ϕ, y = a\* sin3ϕ

ρ,ϕ- полярные координаты, x,y – декартовы координаты t – независимый параметр.

a,b, k,A,B, - константы, значения которых выбираются пользователем (вводятся в окне программы). a,b>0

Обеспечить автоматическое масштабирование и центрирование кривой при изменении размеров окна.

**Решение задачи**

По условию переведём координаты в полярные координаты: x = a\* cos3ϕ, y = a\* sin3ϕ.  
В итоге параметризация происходит по ϕ: 2π делится на заданное пользователем число и рисовка происходит с соответствующим шагом. Также здесь реализованы сдвиг, поворот графика, автомасштаб/масштаб и рисовка осей. Для корректного отображения к соответствующим координатам прибавлены половины от высоты/ширины окна.

**Листинг программы**

*//#define UseOpenGL // Раскомментировать для использования OpenGL*

#if (!UseOpenGL)

using Device = CGLabPlatform.GDIDevice;

using DeviceArgs = CGLabPlatform.GDIDeviceUpdateArgs;

#else

using Device = CGLabPlatform.OGLDevice;

using DeviceArgs = CGLabPlatform.OGLDeviceUpdateArgs;

using SharpGL;

#endif

using System;

using System.Linq;

using System.Drawing;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Collections.Generic;

using CGLabPlatform;

using CGApplication = MyApp;

public abstract class MyApp: CGApplicationTemplate<CGApplication, Device, DeviceArgs> {

*// TODO: Добавить свойства, поля*

public const int **width** = 300; *// VSPanelWidth ширина поля с кнопками*

[DisplayNumericProperty(Default: 100, Increment: 1, Minimum: 0, Maximum: 1000, Name: "Постоянная А ")]

public abstract double A { get; set; }

[DisplayNumericProperty(Default: 1000, Increment: 1, Minimum: 1, Maximum: 10000, Name: "Аппроксимация ")]

public abstract int VertexCount { get; set; }

[DisplayNumericProperty(Default: 2, Increment: 0.5, Minimum: 0.1, Maximum: 10, Name: "Zoom ")]

public abstract double zoom { get; set; }

[DisplayNumericProperty(Default: 0, Increment: 0.05, Minimum: 0, Maximum: 2, Name: "Коэффициент поворота\*(pi) ")]

public abstract double angel { get; set; }

protected DVector2 Rotation(DVector2 point)

{

DVector2 resPoint;

*//X = (x — x0) \* cos(alpha) — (y — y0) \* sin(alpha) + x0;*

*//Y = (x — x0) \* sin(alpha) + (y — y0) \* cos(alpha) + y0;*

var cosA = Math.Cos(angel \* Math.**PI**);

var sinA = Math.Sin(angel \* Math.**PI**);

resPoint.X = (point.X - CenterFig.X) \* cosA - (point.Y - CenterFig.Y) \* sinA + CenterFig.X;

resPoint.Y = (point.X - CenterFig.X) \* sinA + (point.Y - CenterFig.Y) \* cosA + CenterFig.Y;

return resPoint;

}

public DVector2 Center;

public DVector2 CenterFig;

protected void CalculateCenter() *// вычисление точки центра окна и точки центра фигуры*

{

Center = new DVector2(((base.MainWindow.Width - **width**) / 2) , (base.MainWindow.Height / 2) );

CenterFig = new DVector2( Shift.X, - Shift.Y);

CenterFig.X \*= (base.MainWindow.Height \* base.MainWindow.Width) / (WindowWidth \* WindowHeight);

CenterFig.Y \*= (base.MainWindow.Height \* base.MainWindow.Width) / (WindowWidth \* WindowHeight);

CenterFig.X += Center.X;

CenterFig.Y += Center.Y;

}

protected double WindowWidth ;

protected double WindowHeight ;

protected DVector2 FromViewToPhysicalSpace(DVector2 point) {*// Преобразование координат из видового пространства в физическое*

double coeff ; *// Коэффициент вычисляющийся из длинны и ширины окна. Используется для корректировки размера граффика*

WindowWidth = 2500;

WindowHeight = 1380;

this.CalculateCenter();

DVector2 result = new DVector2(point.X , -point.Y ); *// В физическом пространстве ось X направлена вправо, а Y направлена вниз*

coeff = (base.MainWindow.Height \* base.MainWindow.Width) / (WindowWidth \* WindowHeight) ;

result.X \*= coeff;

*//coeff = (base.MainWindow.Height \* base.MainWindow.Width)/ (WindowHeight \* WindowWidth);*

result.Y \*= coeff;

result.X += Center.X;

result.Y += Center.Y;

return result;

}

public abstract DVector2 Shift { get; set; }

public double delta;

protected override void OnMainWindowLoad(object sender, EventArgs args)

{

*// Созданное приложение имеет два основных элемента управления:*

*// base.RenderDevice - левая часть экрана для рисования*

*// base.ValueStorage - правая панель для отображения и редактирования свойств*

*// Пример изменения внешниго вида элементов управления (необязательный код)*

base.RenderDevice.BufferBackCol = 0x20;

base.ValueStorage.Font = new Font("Arial", 12f);

base.ValueStorage.ForeColor = Color.Firebrick;

base.ValueStorage.RowHeight = 30;

base.ValueStorage.BackColor = Color.BlanchedAlmond;

base.MainWindow.BackColor = Color.DarkGoldenrod;

base.ValueStorage.RightColWidth = 50;

base.VSPanelWidth = **width**;

base.VSPanelLeft = true;

base.MainWindow.Size = new Size(2500, 1380);

base.MainWindow.StartPosition = FormStartPosition.**Manual**;

base.MainWindow.Location = Point.Empty;

base.RenderDevice.GraphicsHighSpeed = false;

*// Реализация управления мышкой с зажатыми левой и правой кнопкой мыши*

base.RenderDevice.MouseMoveWithRightBtnDown += (s, e)

=> Shift += 10 \* new DVector2(e.MovDeltaX, -e.MovDeltaY);

base.RenderDevice.MouseMoveWithLeftBtnDown += (s, e)

=> Shift += 0.5 \* new DVector2(e.MovDeltaX, -e.MovDeltaY);

*// Реализация управления клавиатурой*

RenderDevice.HotkeyRegister(Keys.**Up**, (s, e) => Shift += DVector2.UnitY);

RenderDevice.HotkeyRegister(Keys.**Down**, (s, e) => Shift -= DVector2.UnitY);

RenderDevice.HotkeyRegister(Keys.**Left**, (s, e) => Shift -= DVector2.UnitX);

RenderDevice.HotkeyRegister(Keys.**Right**, (s, e) => Shift += DVector2.UnitX);

RenderDevice.HotkeyRegister(KeyMod.**Shift**, Keys.**Up**, (s, e) => Shift +=10\*DVector2.UnitY);

RenderDevice.HotkeyRegister(KeyMod.**Shift**, Keys.**Down**, (s, e) => Shift -=10\*DVector2.UnitY);

RenderDevice.HotkeyRegister(KeyMod.**Shift**, Keys.**Left**, (s, e) => Shift -=10\*DVector2.UnitX);

RenderDevice.HotkeyRegister(KeyMod.**Shift**, Keys.**Right**, (s, e) => Shift +=10\*DVector2.UnitX);

}

protected void DrawScaleOY(DeviceArgs e){ *// разметка оси Y*

var tmpCenterL = new DVector2(-10, 0);

var tmpCenterR = new DVector2(10, 0);

for (int i = 0; i < 5; i++)

{

tmpCenterL.Y += 30;

tmpCenterR.Y += 30;

e.Surface.DrawLine(Color.Red.ToArgb(), FromViewToPhysicalSpace(tmpCenterL \*zoom + Shift), FromViewToPhysicalSpace(tmpCenterR \*zoom + Shift));

}

tmpCenterL.Y = 0;

tmpCenterR.Y = 0;

for (int i = 0; i < 5; i++)

{

tmpCenterL.Y -= 30;

tmpCenterR.Y -= 30;

e.Surface.DrawLine(Color.Red.ToArgb(), FromViewToPhysicalSpace(tmpCenterL \*zoom + Shift), FromViewToPhysicalSpace(tmpCenterR \*zoom + Shift));

}

}

protected void DrawScaleOX(DeviceArgs e){*// разметка оси Х*

var tmpCenterL = new DVector2(0, -10);

var tmpCenterR = new DVector2(0, 10);

for (int i = 0; i < 5; i++){

tmpCenterL.X += 30;

tmpCenterR.X += 30;

e.Surface.DrawLine(Color.Red.ToArgb(), FromViewToPhysicalSpace(tmpCenterL \*zoom + Shift), FromViewToPhysicalSpace(tmpCenterR \*zoom + Shift));

}

tmpCenterL.X = 0;

tmpCenterR.X = 0;

for (int i = 0; i < 5; i++){

tmpCenterL.X -= 30;

tmpCenterR.X -= 30;

e.Surface.DrawLine(Color.Red.ToArgb(), FromViewToPhysicalSpace(tmpCenterL \*zoom + Shift), FromViewToPhysicalSpace(tmpCenterR \*zoom + Shift));

}

}

protected override void OnDeviceUpdate(object s, DeviceArgs e) {

*// TODO: Отрисовка и обновление*

double step = 2 \* Math.**PI** / VertexCount;

double angle = 0;

double X, Y;

*//создаем OX, OY и разметку*

DVector2 mOX = new DVector2(-800, 0);

DVector2 OX = new DVector2(800, 0);

DVector2 mOY = new DVector2(0, -800);

DVector2 OY = new DVector2(0, 800);

e.Surface.DrawLine(Color.Red.ToArgb(), FromViewToPhysicalSpace(mOX \*zoom + Shift), FromViewToPhysicalSpace(OX \*zoom + Shift));

e.Surface.DrawLine(Color.Red.ToArgb(), FromViewToPhysicalSpace(mOY \*zoom + Shift), FromViewToPhysicalSpace(OY \*zoom + Shift));

this.DrawScaleOY(e);

this.DrawScaleOX(e);

*//создали OX, OY и разметку*

List<DVector2> points = new List<DVector2>();

while (angle < 2 \* Math.**PI**) { *// просчет точек графика*

X = A \* Math.Pow(Math.Cos(angle),3) \*zoom;

Y = A \* Math.Pow (Math.Sin(angle),3) \*zoom;

points.Add(new DVector2(X, Y)+Shift);

angle += step;

}

X = A \* Math.Pow(Math.Cos(angle),3) \*zoom;

Y = A \* Math.Pow (Math.Sin(angle),3) \*zoom;

points.Add(new DVector2(X, Y)+Shift);

for (int i = 1; i < points.Count; ++i) { *// рисуем график*

e.Surface.DrawLine(Color.LawnGreen.ToArgb(), Rotation(FromViewToPhysicalSpace(points[i])), Rotation(FromViewToPhysicalSpace(points[i - 1])));

}

}

}

public abstract class AppMain : CGApplication

{ [STAThread] static void Main() { RunApplication(); } }

**Вывод**

В результате выполнения данной лабораторной работы я познакомился с возможностью отрисовки 2D изображений кривых посредством вызова методов рисования отрезков в форме приложения. Также я научился применять такие простые аффинные преобразования как сдвиг, поворот и масштабирование.

Лабораторная работа №2 по курсу «Компьютерная графика»

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

**Постановка задачи**

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

Вариант: 2. Правильный октаэдр

**Решение задачи**

Формироваться заданная вариантом фигура будет следующим образом. Будем считать, что это две пирамиды в основании которых квадрат, они склеины основаниями. Вершина одной пирамиды находится в точке (0, 0, PyrHeight), вершина другой пирамиды находится в точке (0, 0, -PyrHeight). Для задания вершины пирамиды достаточно взять координату центра окружности и к значению z прибавить нужное значение высоты. Тут стоит обратить внимание на то, что для каждого полигона, составляющего фигуру, нужно посчитать нормаль к плоскости для определения при отрисовки видимых и невидимых граней соответственно. Для возможного проведения аффинных преобразований путем матричного перемножения необходимо считать все точки четырехмерными - появляется w координата, всегда равная 1 (для векторов - 0). После такого введения задание преобразующих матриц не составит труда, все их можно перемножить для получения итоговой матрицы одного сложного преобразования. Отдельную матрицу преобразований также требуется получить и для нормалей, чтобы каждый раз их не пересчитывать при любом изменении положения фигуры. После перевода исходных координат фигуры из видового пространства в мировое путем матричного перемножения также требуется перевести полученные координаты в физическую систему окна отрисовки - для этого сперва производим проецирование фигуры на плоскость путем деления x и y координаты каждой точки на w, а далее делаем все тоже самое, что было проделано в ЛР 1 - определяем коэффициент преобразования для каждой координаты в соответствии с размерами окна и области отрисовки.

**Листинг программы**

//#define UseOpenGL // Раскомментировать для использования OpenGL

#if (!UseOpenGL)

using Device = CGLabPlatform.GDIDevice;

using DeviceArgs = CGLabPlatform.GDIDeviceUpdateArgs;

#else

using Device = CGLabPlatform.OGLDevice;

using DeviceArgs = CGLabPlatform.OGLDeviceUpdateArgs;

using SharpGL;

#endif

using System;

using System.Linq;

using System.Drawing;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Collections.Generic;

using System.Diagnostics;

using CGLabPlatform;

using CGApplication = MyApp;

using System.ComponentModel;

public abstract class MyApp: CGApplicationTemplate<CGApplication, Device, DeviceArgs>

{

public const int width = 400; // VSPanelWidth ширина поля с кнопками

public class Vertex

{

private List<Polygon> polygons;

public Vertex() { }

public Vertex(double[] elements)

{

Debug.Assert(elements != null && elements.Length == 4);

this.Point\_InLocalSpace.X = elements[0];

this.Point\_InLocalSpace.Y = elements[1];

this.Point\_InLocalSpace.Z = elements[2];

this.Point\_InLocalSpace.W = elements[3];

}

public DVector4 Point\_InLocalSpace;

public DVector4 Point\_InWorldSpace;

}

public class Polygon

{

public Vertex[] vertecies;

public DVector4 Normal\_InLocalSpace;

public DVector4 Normal\_InWorldSpace;

public Polygon() { }

public Polygon(Vertex[] elements)

{

Debug.Assert(elements != null && elements.Length == 3);

vertecies = elements;

}

public bool IsVisible;

public int Color;

}

// TODO: Добавить свойства, поля

private DMatrix4 \_PointTransform;

private Commands \_Commands = Commands.FigureChange;

private DVector3 \_Rotation;

private DVector3 \_Offset;

private DVector3 \_Scale;

[Flags]

private enum Commands : int

{

None = 0,

Transform = 1 << 0,

FigureChange = 1 << 1

}

[DisplayNumericProperty(Default: new[] { 1d, 0, 0, 0, 0, 1d, 0, 0, 0, 0, 1d, 0, 0, 0, 0, 1d }, 0.01, 4, null)]

public DMatrix4 PointTransform

{

get { return \_PointTransform; }

set {

if (value == \_PointTransform)

return;

\_PointTransform = value;

\_Commands |= Commands.Transform;

OnPropertyChanged();

}

}

[DisplayNumericProperty(Default: 100, Increment: 0.1, Minimum: 1, Name: "Ширина фигуры")]

public double BaseRadius{

get { return Get<double>(); }

set

{

if (Set<double>(value))

{

\_Commands |= Commands.FigureChange;

}

}

}

[DisplayNumericProperty(Default: 100, Increment: 0.1, Minimum: 1, Name: "Высота фигуры")]

public double PyrHeight{

get { return Get<double>(); }

set

{

if (Set<double>(value))

{

\_Commands |= Commands.FigureChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 0d, 0d, 0d }, Increment: 0.1, Name: "Смещение")]

public DVector3 Offset

{

get { return \_Offset; }

set {

if (Set<DVector3>(value))

{

\_Offset = value;

UpdateTransformMatrix();

}

}

}

[DisplayNumericProperty(Default: new[] { 0d, 0d, 0d }, Increment: 1, Name: "Поворот")]

public DVector3 Rotation

{

get { return \_Rotation; }

set {

if (Set<DVector3>(value))

{

\_Rotation = value;

if (\_Rotation.X >= 360) \_Rotation.X -= 360;

if (\_Rotation.Y >= 360) \_Rotation.Y -= 360;

if (\_Rotation.Z >= 360) \_Rotation.Z -= 360;

if (\_Rotation.X < 0) \_Rotation.X += 360;

if (\_Rotation.Y < 0) \_Rotation.Y += 360;

if (\_Rotation.Z < 0) \_Rotation.Z += 360;

UpdateTransformMatrix();

}

}

}

[DisplayNumericProperty(Default: new[] { 20d, 20d, 20d }, Increment: 0.1, Name: "Масштаб")]

public DVector3 Scale

{

get { return \_Scale; }

set

{

\_Scale = value;

if (Set<DVector3>(value))

{

UpdateTransformMatrix();

}

}

}

public enum Projection

{

[Description("Не задана")] NotSet,

[Description("Вид спереди")] InFront,

[Description("Вид сверху")] Above,

[Description("Вид сбоку")] Sideway,

[Description("Изометрия")] Isometry

}

[DisplayEnumListProperty(Projection.NotSet, Name: "Проекция")]

public Projection CurProjection

{

get { return Get<Projection>(); }

set{

if (Set<Projection>(value)){

if (CurProjection == Projection.Above){

\_Scale.Y = 0;

}

else if (CurProjection == Projection.InFront){

\_Scale.Z = 0;

}

else if (CurProjection == Projection.Sideway){

\_Scale.X = 0;

}

else if (CurProjection == Projection.Isometry){

\_Rotation.X = 35;

\_Rotation.Y = 45;

\_Rotation.Z = 0;

}

UpdateTransformMatrix();

\_Commands |= Commands.Transform;

}

}

}

public enum Visualization{

[Description("Не отображать полигоны")] NoPolygons,

[Description("Одним цветом")] OneColor,

[Description("Случайные цвета")] RandomColor

}

[DisplayEnumListProperty(Visualization.NoPolygons, Name: "Способ отрисовки")]

public abstract Visualization CurVisual { get; set; }

[DisplayCheckerProperty(Default: true, Name: "Каркасная визуализация")]

public abstract bool IsCarcass { get; set; }

public DMatrix4 NormalTransform;

public Vertex[] vertices;

public Polygon[] polygons;

public DVector2 ChangePos;

public DVector2 ChangeAngle;

public DVector2 ViewSize;

public DVector2 Automove;

public DVector2 AutoScale;

private void UpdateTransformMatrix(){

\_PointTransform = new DMatrix4(new double[] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 });

/\* Смещение \*/

DMatrix4 offset = new DMatrix4(new double[] { 1, 0, 0, \_Offset.X, 0, 1, 0, \_Offset.Y,0, 0, 1, \_Offset.Z , 0, 0, 0, 1 });

\_PointTransform \*= offset;

/\* Поворот \*/

// По оси X

DMatrix4 x\_rotate = new DMatrix4(new double[] {1, 0, 0, 0,

0, Math.Cos(Math.PI / 180 \* \_Rotation.X), Math.Sin(Math.PI / 180 \*\_Rotation.X), 0,

0, -Math.Sin(Math.PI / 180 \* \_Rotation.X), Math.Cos(Math.PI / 180 \* \_Rotation.X), 0, 0, 0, 0, 1 });

DMatrix4 y\_rotate = new DMatrix4(new double[] {Math.Cos(Math.PI / 180 \* \_Rotation.Y), 0, Math.Sin(Math.PI / 180 \* \_Rotation.Y), 0, 0, 1, 0, 0, -Math.Sin(Math.PI / 180 \* \_Rotation.Y), 0, Math.Cos(Math.PI / 180 \* \_Rotation.Y), 0, 0, 0, 0, 1 });

DMatrix4 z\_rotate = new DMatrix4(new double[] {Math.Cos(Math.PI / 180 \* \_Rotation.Z), -Math.Sin(Math.PI / 180 \* \_Rotation.Z), 0, 0, Math.Sin(Math.PI / 180 \* \_Rotation.Z), Math.Cos(Math.PI / 180 \* \_Rotation.Z), 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 });

\_PointTransform \*= x\_rotate;

\_PointTransform \*= y\_rotate;

\_PointTransform \*= z\_rotate;

/\* Маштабирование \*/

\_PointTransform \*= new DMatrix4(new double[] {\_Scale.X, 0, 0, 0, 0, \_Scale.Y, 0, 0,0, 0, \_Scale.Z, 0 ,0, 0, 0, 1 });

\_Commands |= Commands.Transform;

}

public void Create()

{

int VertexSize = 6;

int PolygonSize = 8;

vertices = new Vertex[VertexSize]; //6

polygons = new Polygon[PolygonSize]; //8

var random = new Random();

for (int i = 0; i < vertices.Length; ++i){

vertices[i] = new Vertex();

}

for (int i = 0; i < polygons.Length; ++i){

polygons[i] = new Polygon();

polygons[i].Color = random.Next();

}

Generate();

}

public void Generate(){

int numPoint = 5;

for (int i = 0; i <numPoint - 1; ++i){

vertices[i].Point\_InLocalSpace.X = BaseRadius \* Math.Cos(2 \* i \* Math.PI / (numPoint - 1));

vertices[i].Point\_InLocalSpace.Y = BaseRadius \* Math.Sin(2 \* i \* Math.PI / (numPoint - 1));

vertices[i].Point\_InLocalSpace.Z = 0.01;

vertices[i].Point\_InLocalSpace.W = 1;

}

vertices[numPoint - 1].Point\_InLocalSpace.X = 0;

vertices[numPoint - 1].Point\_InLocalSpace.Y = 0;

vertices[numPoint - 1].Point\_InLocalSpace.Z = -1\*PyrHeight ;

vertices[numPoint - 1].Point\_InLocalSpace.W = 1;

vertices[numPoint].Point\_InLocalSpace.X = 0;

vertices[numPoint].Point\_InLocalSpace.Y = 0;

vertices[numPoint].Point\_InLocalSpace.Z = PyrHeight;

vertices[numPoint].Point\_InLocalSpace.W = 1;

polygons[0].vertecies = new []{vertices[numPoint - 1], vertices[0], vertices[1] };

polygons[1].vertecies = new []{vertices[numPoint - 1], vertices[1], vertices[2] };

polygons[2].vertecies = new []{vertices[numPoint - 1], vertices[2], vertices[3] };

polygons[3].vertecies = new []{vertices[numPoint - 1], vertices[3], vertices[0] };

polygons[4].vertecies = new []{vertices[1 ], vertices[0], vertices[numPoint] };

polygons[5].vertecies = new []{vertices[2 ], vertices[1], vertices[numPoint] };

polygons[6].vertecies = new []{vertices[3 ], vertices[2], vertices[numPoint] };

polygons[7].vertecies = new []{vertices[0 ], vertices[3], vertices[numPoint] };

foreach (var p in polygons){

DVector4 first = new DVector4(new double[] {p.vertecies[1].Point\_InLocalSpace.X - p.vertecies[0].Point\_InLocalSpace.X,

p.vertecies[1].Point\_InLocalSpace.Y - p.vertecies[0].Point\_InLocalSpace.Y,

p.vertecies[1].Point\_InLocalSpace.Z - p.vertecies[0].Point\_InLocalSpace.Z,

p.vertecies[1].Point\_InLocalSpace.W - p.vertecies[0].Point\_InLocalSpace.W});

DVector4 second = new DVector4(new double[] {p.vertecies[2].Point\_InLocalSpace.X - p.vertecies[0].Point\_InLocalSpace.X,

p.vertecies[2].Point\_InLocalSpace.Y - p.vertecies[0].Point\_InLocalSpace.Y,

p.vertecies[2].Point\_InLocalSpace.Z - p.vertecies[0].Point\_InLocalSpace.Z,

p.vertecies[2].Point\_InLocalSpace.W - p.vertecies[0].Point\_InLocalSpace.W});

p.Normal\_InLocalSpace = first \* second;

p.Normal\_InLocalSpace.Normalize();

}

}

protected DVector2 FromViewToPhysicalSpace(Vertex point){

DVector2 result = new DVector2();

result.X = point.Point\_InWorldSpace.X / point.Point\_InWorldSpace.W;

result.Y = point.Point\_InWorldSpace.Y / point.Point\_InWorldSpace.W;

result.X = result.X \* AutoScale.X + Automove.X; // Преобразование координат из видового пространства в физическое

result.Y = result.Y \* AutoScale.Y + Automove.Y;

return result;

}

protected override void OnMainWindowLoad(object sender, EventArgs args){

// TODO: Инициализация данных

base.RenderDevice.BufferBackCol = 0x20;

base.ValueStorage.Font = new Font("Arial", 12f);

base.ValueStorage.ForeColor = Color.Firebrick;

base.ValueStorage.RowHeight = 30;

base.ValueStorage.BackColor = Color.BlanchedAlmond;

base.MainWindow.BackColor = Color.DarkGoldenrod;

base.ValueStorage.RightColWidth = 50;

base.VSPanelWidth = width;

base.VSPanelLeft = true;

base.MainWindow.Size = new Size(2500, 1380);

base.MainWindow.StartPosition = FormStartPosition.Manual;

base.MainWindow.Location = Point.Empty;

base.RenderDevice.GraphicsHighSpeed = false;

base.RenderDevice.BufferBackCol = 0x20;

base.RenderDevice.MouseMoveWithRightBtnDown += (s, e)

=> Offset += new DVector3(0.35\*Math.Abs(\_Scale.X)\*e.MovDeltaX, 0.35\*Math.Abs(\_Scale.Y)\*e.MovDeltaY, 0);

base.RenderDevice.MouseMoveWithLeftBtnDown += (s, e)

=> Rotation += new DVector3(0.1 \* e.MovDeltaY, 0.1 \* e.MovDeltaX , 0);

base.RenderDevice.MouseWheel += (s, e) => Scale += new DVector3(0.05 \* e.Delta, 0.05 \* e.Delta, 0.05 \* e.Delta);

Create();

}

protected override void OnDeviceUpdate(object s, DeviceArgs e)

{

if (0 != ((int)\_Commands & (int)Commands.FigureChange)){

\_Commands ^= Commands.FigureChange;

Generate();

}

/\* Обновление значений, использующихся для перевода в физ. с. к. \*/

var x\_min = vertices.Min(p => (p.Point\_InLocalSpace.X / p.Point\_InLocalSpace.Z));

var x\_max = vertices.Max(p => (p.Point\_InLocalSpace.X / p.Point\_InLocalSpace.Z));

var y\_min = vertices.Min(p => (p.Point\_InLocalSpace.Y / p.Point\_InLocalSpace.Z));

var y\_max = vertices.Max(p => (p.Point\_InLocalSpace.Y / p.Point\_InLocalSpace.Z));

ViewSize.X = x\_max - x\_min;

ViewSize.Y = y\_max - y\_min;

AutoScale.X = .9 \* e.Width / ViewSize.X;

AutoScale.Y = .9 \* e.Heigh / ViewSize.Y;

AutoScale.X = AutoScale.Y = Math.Min(AutoScale.X, AutoScale.Y);

Automove.X = e.Width / 2 - (x\_min + x\_max) / 2 \* AutoScale.X;

Automove.Y = e.Heigh / 2 - (y\_min + y\_max) / 2 \* AutoScale.Y;

if (0 != ((int) \_Commands & (int) Commands.Transform)) {

\_Commands ^= Commands.Transform;

// Пересчет преобразования вектора нормали

NormalTransform = DMatrix3.NormalVecTransf(PointTransform);

foreach (var v in vertices){

v.Point\_InWorldSpace = PointTransform \* v.Point\_InLocalSpace;

}

foreach (var p in polygons){

p.Normal\_InWorldSpace = NormalTransform \* p.Normal\_InLocalSpace;

p.IsVisible = p.Normal\_InWorldSpace.Z < 0;

}

polygons.OrderBy(p => Math.Min(p.vertecies[0].Point\_InWorldSpace.Z, Math.Min(p.vertecies[1].Point\_InWorldSpace.Z, p.vertecies[2].Point\_InWorldSpace.Z)));

}

foreach (var p in polygons){

if (!p.IsVisible)

continue;

if (CurVisual == Visualization.OneColor){

e.Surface.DrawTriangle(Color.YellowGreen.ToArgb(), FromViewToPhysicalSpace(p.vertecies[0]), FromViewToPhysicalSpace(p.vertecies[1]),

FromViewToPhysicalSpace(p.vertecies[2]));

}

else if (CurVisual == Visualization.RandomColor){

e.Surface.DrawTriangle(p.Color, FromViewToPhysicalSpace(p.vertecies[0]), FromViewToPhysicalSpace(p.vertecies[1]),

FromViewToPhysicalSpace(p.vertecies[2]));

}

if (IsCarcass)

{

e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[0]), FromViewToPhysicalSpace(p.vertecies[1]));

e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[1]), FromViewToPhysicalSpace(p.vertecies[2]));

e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[2]), FromViewToPhysicalSpace(p.vertecies[0]));

}

}

}

}

public abstract class AppMain : CGApplication

{ [STAThread] static void Main() { RunApplication(); } }

**Вывод**

В результате выполнения данной лабораторной работы я познакомился с возможностью отрисовки 3D изображений простых фигур посредством вызова методов рисования отрезков в форме приложения. Также я научился применять сложные аффинные преобразования в трехмерном пространстве путем матричного произведения, проецировать фигуры в рабочую плоскость - выполнять изометрические и видовые проекции.

Лабораторная работа №3 по курсу «Компьютерная графика»

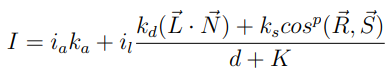
Тема: Основы построения фотореалистичных изображений.

**Постановка задачи**

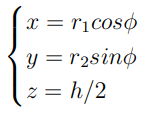
Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.  
Варипнт: 3. Шар

**Решение задачи**

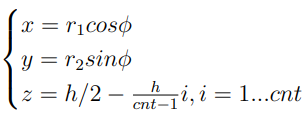
В лабораторной работе требуется реализовать плоскую модель затенения и модель затенения по Гуро. Различия между ними заключаются в том, что первая раскрашивает каждый полигон в один соответствующий цвет, в то время как модель Гуро интерполирует значения цвета, посчитанного для каждой вершины, составляющей полигон. Значения интенсивности для вершин/полигонов будут определяться по следующей формуле, которая объединяет интенсивности диффузной, рассеянной и зеркальной составляющих:



Для задания координат вершин прямого эллиптического цилиндра будем считать его центр масс также и центром видовой системы координат. Далее все очень просто - имеем два уравнения для эллиптических оснований вида:



где r1, r2 - радиусы полуосей, h - высота цилиндра, при задании нижнего основания считаем z = −h/2. Для формирования боковых граней также будем использовать уравнения эллипса, только периодически смещаясь по z координате, задавая тем самым последовательные "этажи"из граней:



где cnt - число этажей, это значение аппроксимации будет задаваться пользователем в программе.

**Листинг программы**

using Device = CGLabPlatform.OGLDevice;

using DeviceArgs = CGLabPlatform.OGLDeviceUpdateArgs;

using SharpGL;

using System;

using System.Linq;

using System.Drawing;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Collections.Generic;

using System.Diagnostics;

using CGLabPlatform;

using CGApplication = MyApp;

using System.ComponentModel;

public abstract class MyApp : CGApplicationTemplate<CGApplication, Device, DeviceArgs>{

#region Классы

public class Vertex{

public Vertex() { }

public Vertex(double[] elements){

Debug.Assert(elements != null && elements.Length == 4);

this.Point\_InLocalSpace.X = elements[0];

this.Point\_InLocalSpace.Y = elements[1];

this.Point\_InLocalSpace.Z = elements[2];

this.Point\_InLocalSpace.W = elements[3];

polygons = new List<Polygon>();

}

public DVector4 Point\_InLocalSpace;

public DVector4 Point\_InWorldSpace;

public List<Polygon> polygons;

public Color LightColor;

}

public class Polygon{

public Vertex[] vertecies;

public DVector4 Normal\_InLocalSpace;

public DVector4 Normal\_InWorldSpace;

public Polygon() { }

public Polygon(Vertex[] elements, int randomColor) {

Debug.Assert(elements != null && elements.Length == 3);

vertecies = elements;

foreach (var v in elements){

v.polygons.Add(this);

}

RandomColor = randomColor;

}

public bool IsVisible;

public int RandomColor;

public Color LightColor;

}

#endregion

// TODO: Добавить свойства, поля

private DMatrix4 \_PointTransform;

private Commands \_Commands = Commands.FigureChange;

private DVector3 \_Rotation;

private DVector3 \_Offset;

private DVector3 \_Scale;

[Flags]

private enum Commands : int{

None = 0,

Transform = 1 << 0,

FigureChange = 1 << 1,

NewFigure = 1 << 2,

ShadingChange = 1 << 3

}

#region Свойства

[DisplayNumericProperty(Default: new[] { 1d, 0, 0, 0, 0, 1d, 0, 0, 0, 0, 1d, 0,0, 0, 0, 1d }, 0.01, 4, null)]

public DMatrix4 PointTransform {

get { return \_PointTransform; }

set{

if (value == \_PointTransform)

return;

\_PointTransform = value;

\_Commands |= Commands.Transform;

OnPropertyChanged();

}

}

[DisplayNumericProperty(Default: 1, Increment: 0.1, Minimum: 1, Name: "Радиус")]

public double Radius {

get { return Get<double>(); }

set{

if (Set<double>(value)){

\_Commands |= Commands.FigureChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 12d, 16d }, Minimum: 4d, Increment: 2d, Name: "Апроксимация")]

public DVector2 Approximation{

get { return Get<DVector2>(); }

set{

if (Set<DVector2>(value)){

\_Commands |= Commands.NewFigure;

}

}

}

public int approx0, approx1;

[DisplayNumericProperty(Default: new[] { 0d, 0d, 0d }, Increment: 0.1, Name: "Смещение")]

public DVector3 Offset{

get { return \_Offset; }

set{

if (Set<DVector3>(value)){

\_Offset = value;

UpdateTransformMatrix();

}

}

}

[DisplayNumericProperty(Default: new[] { 0d, 0d, 0d }, Increment: 1, Name: "Поворот")]

public DVector3 Rotation{

get { return \_Rotation; }

set{

if (Set<DVector3>(value)){

\_Rotation = value;

if (\_Rotation.X >= 360) \_Rotation.X -= 360;

if (\_Rotation.Y >= 360) \_Rotation.Y -= 360;

if (\_Rotation.Z >= 360) \_Rotation.Z -= 360;

if (\_Rotation.X < 0) \_Rotation.X += 360;

if (\_Rotation.Y < 0) \_Rotation.Y += 360;

if (\_Rotation.Z < 0) \_Rotation.Z += 360;

UpdateTransformMatrix();

}

}

}

[DisplayNumericProperty(Default: new[] { 1d, 1d, 1d }, Minimum: 0.1d, Increment: 0.1, Name: "Масштаб")]

public DVector3 Scale{

get { return \_Scale; }

set{

\_Scale = value;

if (Set<DVector3>(value)){

UpdateTransformMatrix();

}

}

}

public enum Projection{

[Description("Не задана")] NotSet,

[Description("Вид спереди")] InFront,

[Description("Вид сверху")] Above,

[Description("Вид сбоку")] Sideway,

[Description("Изометрия")] Isometry

}

[DisplayEnumListProperty(Projection.NotSet, Name: "Проекция")]

public Projection CurProjection{

get { return Get<Projection>(); }

set{

if (Set<Projection>(value)){

if (CurProjection == Projection.Above){

\_Scale.Y = 0;

}

else if (CurProjection == Projection.InFront){

\_Scale.Z = 0;

}

else if (CurProjection == Projection.Sideway){

\_Scale.X = 0;

}

else if (CurProjection == Projection.Isometry){

\_Rotation.X = 35;

\_Rotation.Y = 45;

\_Rotation.Z = 0;

}

UpdateTransformMatrix();

\_Commands |= Commands.Transform;

}

}

}

public enum Visualization{

[Description("Не отображать полигоны")] NoPolygons,

[Description("Одним цветом")] OneColor,

[Description("Случайные цвета")] RandomColor,

[Description("Плоское затенение")] FlatShading,

[Description("Метод затенения Гуро")] GuroShading

}

[DisplayEnumListProperty(Visualization.GuroShading, Name: "Способ отрисовки")]

public Visualization CurVisual{

get { return Get<Visualization>(); }

set{

if (Set<Visualization>(value)){

if (value == Visualization.FlatShading || value == Visualization.GuroShading){

\_Commands |= Commands.ShadingChange;

}

}

}

}

[DisplayCheckerProperty(Default: false, Name: "Каркасная визуализация")]

public abstract bool IsCarcass { get; set; }

[DisplayCheckerProperty(Default: true, Name: "Отображать источник света")]

public abstract bool IsLightSource { get; set; }

[DisplayNumericProperty(Default: new[] { 0.68d, 0.85d, 0.90d }, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Цвет материала")]

public DVector3 MaterialColor{

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 0.14d, 0.14d, 0.20d }, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Ka материала")]

public DVector3 Ka\_Material{

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 1d, 1d, 0.54d }, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Kd материала")]

public DVector3 Kd\_Material {

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 0.21d, 0.21d, 1d }, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Ks материала")]

public DVector3 Ks\_Material{

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: 1d, Minimum: 0.01d, Maximum: 100d, Increment: 0.01d, Name: "p материала")]

public double P\_Material{

get { return Get<double>(); }

set{

if (Set<double>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 1d, 1d, 1d }, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Ia освещения")]

public DVector3 Ia\_Material{

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 1d, 0.5d, 0d }, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Il освещения")]

public DVector3 Il\_Material{

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

[DisplayNumericProperty(Default: new[] { 2.5d, 0.5d, 2d }, Increment: 0.01d, Name: "Pos освещения")]

public DVector3 LightPos

{

get { return Get<DVector3>(); }

set{

if (Set<DVector3>(value))

{

\_Commands |= Commands.Transform;

}

}

}

public DVector4 LightPos\_InWorldSpace;

[DisplayNumericProperty(Default: new[] { 0.1d, 0.35d }, Minimum: 0.01d, Maximum: 100d, Increment: 0.01d, Name: "md, mk")]

public DVector2 Parameters{

get { return Get<DVector2>(); }

set {

if (Set<DVector2>(value)){

\_Commands |= Commands.ShadingChange;

}

}

}

public DMatrix4 NormalTransform;

public Vertex[] vertices;

public Polygon[] polygons;

public DVector2 ChangePos;

public DVector2 ChangeAngle;

public DVector2 ViewSize;

public DVector2 Automove;

public DVector2 AutoScale;

public DVector2 Center;

#endregion

#region Методы

private void UpdateTransformMatrix(){

\_PointTransform = new DMatrix4(new double[] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 });

/\* Смещение \*/

DMatrix4 offset = new DMatrix4(new double[] { 1, 0, 0, \_Offset.X, 0, 1, 0, \_Offset.Y, 0, 0, 1, \_Offset.Z, 0, 0, 0, 1 });

\_PointTransform \*= offset;

/\* Поворот \*/

// По оси X

DMatrix4 x\_rotate = new DMatrix4(new double[] {1, 0, 0, 0,

0, Math.Cos(Math.PI / 180 \* \_Rotation.X), -Math.Sin(Math.PI / 180 \* \_Rotation.X), 0,

0, Math.Sin(Math.PI / 180 \* \_Rotation.X), Math.Cos(Math.PI / 180 \* \_Rotation.X), 0, 0, 0, 0, 1 });

DMatrix4 y\_rotate = new DMatrix4(new double[] {Math.Cos(Math.PI / 180 \* \_Rotation.Y), 0, -Math.Sin(Math.PI / 180 \* \_Rotation.Y), 0,0, 1, 0, 0, Math.Sin(Math.PI / 180 \* \_Rotation.Y), 0, Math.Cos(Math.PI / 180 \* \_Rotation.Y), 0, 0, 0, 0, 1 });

DMatrix4 z\_rotate = new DMatrix4(new double[] {Math.Cos(Math.PI / 180 \* \_Rotation.Z), -Math.Sin(Math.PI / 180 \* \_Rotation.Z), 0, 0, Math.Sin(Math.PI / 180 \* \_Rotation.Z), Math.Cos(Math.PI / 180 \* \_Rotation.Z), 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 });

\_PointTransform \*= x\_rotate;

\_PointTransform \*= y\_rotate;

\_PointTransform \*= z\_rotate;

/\* Маштабирование \*/

\_PointTransform \*= new DMatrix4(new double[] {\_Scale.X, 0, 0, 0, 0, \_Scale.Y, 0, 0, 0, 0, \_Scale.Z, 0 , 0, 0, 0, 1 });

\_Commands |= Commands.Transform;

}

public void Create(){

approx0 = (int)Approximation[0];// меридианы

approx1 = (int)Approximation[1]; //параллели

Generate();

}

public void Generate(){

List<List<Vertex>> LLvertices = new List<List<Vertex>>();

List<Polygon> Lpolygons = new List<Polygon>();

double phi = (2 \* Math.PI) / (approx0 ); // горизонтально

double theta = (2 \* Math.PI) / (approx1); // вертикально

double sumphi = 0;

double sumtheta = 0;

int indx = 0;

var random = new Random();

for (int i = 0; i < (approx1 / 2) - 1; i++){

sumtheta += theta;

LLvertices.Add(new List<Vertex>());

for (int j = 0; j < approx0; j++){

sumphi += phi;

LLvertices[LLvertices.Count - 1].Add(new Vertex(new[] { Radius \* Math.Sin(sumtheta) \* Math.Cos(sumphi), Radius \* Math.Sin(sumtheta) \* Math.Sin(sumphi), Radius \* Math.Cos(sumtheta), 1 }));

}

sumphi = 0;

}

LLvertices.Add(new List<Vertex>());

LLvertices[LLvertices.Count - 1].Add(new Vertex(new[] { 0, 0, -Radius, 1})); // i - 1 \_\_\_\_\_\_\_\_

LLvertices.Add(new List<Vertex>());

LLvertices[LLvertices.Count - 1].Add(new Vertex(new[] { 0, 0, Radius, 1})); // i +++++

for (int i = 0; i < LLvertices.Count - 3; i++){

for (int j = 0; j < LLvertices[i].Count - 1; j++){

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[i+1][j+1], LLvertices[i][j + 1], LLvertices[i][j] }, random.Next()));

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[i+1][j], LLvertices[i + 1][j + 1], LLvertices[i][j] }, random.Next()));

}

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[i+1][0], LLvertices[i][0], LLvertices[i][LLvertices[i].Count - 1] }, random.Next()));

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[i+1][LLvertices[i+1].Count - 1], LLvertices[i + 1][0], LLvertices[i][LLvertices[i].Count - 1] }, random.Next()));

}

// верхняя крышка

for (int i = (LLvertices[0].Count - 1); i > 0; i--){

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[0][i], LLvertices[LLvertices.Count - 1][0], LLvertices[0][i - 1] }, random.Next()));

}

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[0][0], LLvertices[LLvertices.Count - 1][0], LLvertices[0][LLvertices[0].Count - 1] }, random.Next()));

// нижняя крышка

for (int i = LLvertices[LLvertices.Count - 3].Count - 1; i > 0; i--){

Lpolygons.Add(new Polygon(new Vertex[] { LLvertices[LLvertices.Count - 3][i - 1], LLvertices[LLvertices.Count - 2][0], LLvertices[LLvertices.Count - 3][i ] }, random.Next() )); //random.Next()

}

Lpolygons.Add(new Polygon(new Vertex[] {LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1], LLvertices[LLvertices.Count - 2][0], LLvertices[LLvertices.Count - 3][0]}, random.Next())); //LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1 - 1]

vertices = new Vertex[(LLvertices.Count - 2) \* LLvertices[0].Count + 2];

polygons = new Polygon[Lpolygons.Count];

for (int i = 0; i < vertices.Length; ++i){

vertices[i] = new Vertex();

}

int art\_it = 0;

foreach (var iter in LLvertices){

foreach (var iter1 in iter){

vertices[art\_it] = iter1;

++art\_it;

}

}

for (int i = 0; i < polygons.Length; ++i){

polygons[i] = new Polygon();

polygons[i] = Lpolygons[i];

}

foreach (var p in polygons) {

DVector4 first = new DVector4(new double[] {p.vertecies[1].Point\_InLocalSpace.X - p.vertecies[0].Point\_InLocalSpace.X,

p.vertecies[1].Point\_InLocalSpace.Y - p.vertecies[0].Point\_InLocalSpace.Y,

p.vertecies[1].Point\_InLocalSpace.Z - p.vertecies[0].Point\_InLocalSpace.Z,

p.vertecies[1].Point\_InLocalSpace.W - p.vertecies[0].Point\_InLocalSpace.W});

DVector4 second = new DVector4(new double[] {p.vertecies[2].Point\_InLocalSpace.X - p.vertecies[0].Point\_InLocalSpace.X,

p.vertecies[2].Point\_InLocalSpace.Y - p.vertecies[0].Point\_InLocalSpace.Y,

p.vertecies[2].Point\_InLocalSpace.Z - p.vertecies[0].Point\_InLocalSpace.Z,

p.vertecies[2].Point\_InLocalSpace.W - p.vertecies[0].Point\_InLocalSpace.W});

p.Normal\_InLocalSpace = first \* second;

p.Normal\_InLocalSpace.Normalize();

}

}

public double[] CalculateIntensity(DVector4 point, DVector4 normal) {

DVector4 L = new DVector4(LightPos\_InWorldSpace.X - point.X, LightPos\_InWorldSpace.Y - point.Y, LightPos\_InWorldSpace.Z - point.Z, 0);

double distance = L.GetLength();

L.Normalize();

/\* Фоновая составляющая \*/

double I\_red = Ia\_Material.X \* Ka\_Material.X;

double I\_green = Ia\_Material.Y \* Ka\_Material.Y;

double I\_blue = Ia\_Material.Z \* Ka\_Material.Z;

/\* Рассеянная составляющая \*/

I\_red += Math.Min(1, Math.Max(0, Il\_Material.X \* (Kd\_Material.X \* DVector4.DotProduct(L, normal)) / (Parameters[0] \* distance + Parameters[1])));

I\_green += Math.Min(1, Math.Max(0, Il\_Material.Y \* (Kd\_Material.Y \* DVector4.DotProduct(L, normal)) / (Parameters[0] \* distance + Parameters[1])));

I\_blue += Math.Min(1, Math.Max(0, Il\_Material.Z \* (Kd\_Material.Z \* DVector4.DotProduct(L, normal)) / (Parameters[0] \* distance + Parameters[1])));

/\* Зеркальная составляющая \*/

if (DVector4.DotProduct(L, normal) > 0){

DVector4 S = new DVector4(Center.X - point.X, Center.Y - point.Y, -1000 - point.Z, 0);

DVector4 R = new DVector4(DVector3.Reflect((DVector3)(-L), (DVector3)normal), 0);

S.Normalize();

R.Normalize();

if (DVector4.DotProduct(R, S) > 0){

I\_red += Math.Min(1, Math.Max(0, Il\_Material.X \* Ks\_Material.X \* Math.Pow(DVector4.DotProduct(R, S), P\_Material) / (Parameters[0] \* distance + Parameters[1])));

I\_green += Math.Min(1, Math.Max(0, Il\_Material.Y \* Ks\_Material.Y \* Math.Pow(DVector4.DotProduct(R, S), P\_Material) / (Parameters[0] \* distance + Parameters[1])));

I\_blue += Math.Min(1, Math.Max(0, Il\_Material.Z \* Ks\_Material.Z \* Math.Pow(DVector4.DotProduct(R, S), P\_Material) / (Parameters[0] \* distance + Parameters[1])));

}

}

I\_red = Math.Min(1, I\_red);

I\_green = Math.Min(1, I\_green);

I\_blue = Math.Min(1, I\_blue);

return new double[] { I\_red, I\_green, I\_blue };

}

public void LightCalculation()

{

if (CurVisual == Visualization.FlatShading){

foreach (var p in polygons){

if (!p.IsVisible) continue;

DVector4 polMiddle = new DVector4(p.vertecies.Sum(v => v.Point\_InWorldSpace.X) / 3, p.vertecies.Sum(v => v.Point\_InWorldSpace.Y) / 3,p.vertecies.Sum(v => v.Point\_InWorldSpace.Z) / 3,1d);

double[] result = CalculateIntensity(polMiddle, p.Normal\_InWorldSpace);

p.LightColor = Color.FromArgb((int)Math.Max(0, Math.Min(255, 255 \* result[0] \* MaterialColor.X)), (int)Math.Max(0, Math.Min(255, 255 \* result[1] \* MaterialColor.Y)),

(int)Math.Max(0, Math.Min(255, 255 \* result[2] \* MaterialColor.Z)));

}

}

else if (CurVisual == Visualization.GuroShading){

foreach (var v in vertices){

DVector4 v\_normal = new DVector4(0, 0, 0, 0);

foreach (var p in v.polygons){

v\_normal += p.Normal\_InWorldSpace;

}

v\_normal.Normalize();

double[] result = CalculateIntensity(v.Point\_InWorldSpace, v\_normal);

v.LightColor = Color.FromArgb((int)Math.Max(0, Math.Min(255, 255 \* result[0] \* MaterialColor.X)), (int)Math.Max(0, Math.Min(255, 255 \* result[1] \* MaterialColor.Y)),

(int)Math.Max(0, Math.Min(255, 255 \* result[2] \* MaterialColor.Z)));

}

}

}

protected DVector2 FromViewToPhysicalSpace(DVector4 point){

DVector2 result = new DVector2();

result.X = point.X / point.W;

result.Y = point.Y / point.W;

result.X = result.X \* AutoScale.X + Automove.X; // Преобразование координат из видового пространства в физическое

result.Y = result.Y \* AutoScale.Y + Automove.Y;

return result;

}

#endregion

protected override void OnMainWindowLoad(object sender, EventArgs args){

// TODO: Инициализация данных

base.RenderDevice.BufferBackCol = 0x20;

base.ValueStorage.Font = new Font("Arial", 12f);

base.ValueStorage.ForeColor = Color.Firebrick;

base.ValueStorage.RowHeight = 30;

base.ValueStorage.BackColor = Color.BlanchedAlmond;

base.MainWindow.BackColor = Color.DarkGoldenrod;

base.ValueStorage.RightColWidth = 50;

base.VSPanelWidth = 400;

base.VSPanelLeft = true;

base.MainWindow.Size = new Size(2500, 1380);

base.MainWindow.StartPosition = FormStartPosition.Manual;

base.MainWindow.Location = Point.Empty;

base.RenderDevice.GraphicsHighSpeed = false;

base.RenderDevice.BufferBackCol = 0x20;

base.RenderDevice.MouseMoveWithRightBtnDown += (s, e)

=> Offset += new DVector3(0.005 \* Math.Abs(\_Scale.X) \* e.MovDeltaX, 0.005 \* Math.Abs(\_Scale.Y) \* e.MovDeltaY, 0);

base.RenderDevice.MouseMoveWithLeftBtnDown += (s, e)

=> Rotation += new DVector3(0.1 \* e.MovDeltaY, 0.1 \* e.MovDeltaX, 0);

base.RenderDevice.MouseWheel += (s, e) => Scale += new DVector3(0.001 \* e.Delta, 0.001 \* e.Delta, 0.001 \* e.Delta);

Create();

}

protected override void OnDeviceUpdate(object s, DeviceArgs e){

if (0 != ((int)\_Commands & (int)Commands.NewFigure)){

\_Commands ^= Commands.NewFigure;

Create();

\_Commands |= Commands.FigureChange;

}

if (0 != ((int)\_Commands & (int)Commands.FigureChange)){

\_Commands ^= Commands.FigureChange;

Generate();

\_Commands |= Commands.Transform;

}

/\* Обновление значений, использующихся для перевода в физ. с. к. \*/

var x\_min = vertices.Min(p => (p.Point\_InLocalSpace.X));

var x\_max = vertices.Max(p => (p.Point\_InLocalSpace.X));

var y\_min = vertices.Min(p => (p.Point\_InLocalSpace.Y));

var y\_max = vertices.Max(p => (p.Point\_InLocalSpace.Y));

ViewSize.X = x\_max - x\_min;

ViewSize.Y = y\_max - y\_min;

Center.X = x\_min + ViewSize.X / 2;

Center.Y = y\_min + ViewSize.Y / 2;

AutoScale.X = .9 \* e.Width / ViewSize.X;

AutoScale.Y = .9 \* e.Heigh / ViewSize.Y;

AutoScale.X = AutoScale.Y = Math.Min(AutoScale.X, AutoScale.Y);

Automove.X = e.Width / 2 - (x\_min + x\_max) / 2 \* AutoScale.X;

Automove.Y = e.Heigh / 2 - (y\_min + y\_max) / 2 \* AutoScale.Y;

if (0 != ((int)\_Commands & (int)Commands.Transform)){

\_Commands ^= Commands.Transform;

// Пересчет преобразования вектора нормали

NormalTransform = DMatrix3.NormalVecTransf(PointTransform);

foreach (var v in vertices){

v.Point\_InWorldSpace = PointTransform \* v.Point\_InLocalSpace;

}

DVector4 LightPosV4 = new DVector4(LightPos.X, LightPos.Y, LightPos.Z, 1);

LightPos\_InWorldSpace = PointTransform \* LightPosV4;

foreach (var p in polygons){

p.Normal\_InWorldSpace = NormalTransform \* p.Normal\_InLocalSpace;

p.Normal\_InWorldSpace.Normalize();

p.IsVisible = p.Normal\_InWorldSpace.Z < 0;

}

polygons.OrderBy(p => Math.Min(p.vertecies[0].Point\_InWorldSpace.Z, Math.Min(p.vertecies[1].Point\_InWorldSpace.Z, p.vertecies[2].Point\_InWorldSpace.Z)));

\_Commands |= Commands.ShadingChange;

}

if (0 != ((int)\_Commands & (int)Commands.ShadingChange)){

\_Commands ^= Commands.ShadingChange;

LightCalculation();

}

foreach (var p in polygons){

if (!p.IsVisible)

continue;

if (CurVisual == Visualization.OneColor) {

e.Surface.DrawTriangle(Color.YellowGreen.ToArgb(), FromViewToPhysicalSpace(p.vertecies[0].Point\_InWorldSpace), FromViewToPhysicalSpace(p.vertecies[1].Point\_InWorldSpace), FromViewToPhysicalSpace(p.vertecies[2].Point\_InWorldSpace));

}

else if (CurVisual == Visualization.RandomColor){

e.Surface.DrawTriangle(p.RandomColor, FromViewToPhysicalSpace(p.vertecies[0].Point\_InWorldSpace),

FromViewToPhysicalSpace(p.vertecies[1].Point\_InWorldSpace),

FromViewToPhysicalSpace(p.vertecies[2].Point\_InWorldSpace));

}

else if (CurVisual == Visualization.FlatShading){

e.Surface.DrawTriangle(p.LightColor.ToArgb(), FromViewToPhysicalSpace(p.vertecies[0].Point\_InWorldSpace),

FromViewToPhysicalSpace(p.vertecies[1].Point\_InWorldSpace),

FromViewToPhysicalSpace(p.vertecies[2].Point\_InWorldSpace));

}

else if (CurVisual == Visualization.GuroShading){

DVector2 v1 = FromViewToPhysicalSpace(p.vertecies[0].Point\_InWorldSpace);

DVector2 v2 = FromViewToPhysicalSpace(p.vertecies[1].Point\_InWorldSpace);

DVector2 v3 = FromViewToPhysicalSpace(p.vertecies[2].Point\_InWorldSpace);

e.Surface.DrawTriangle(p.vertecies[0].LightColor.ToArgb(), v1.X, v1.Y,

p.vertecies[1].LightColor.ToArgb(), v2.X, v2.Y,

p.vertecies[2].LightColor.ToArgb(), v3.X, v3.Y);

}

if (IsCarcass){

e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[0].Point\_InWorldSpace), FromViewToPhysicalSpace(p.vertecies[1].Point\_InWorldSpace));

e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[1].Point\_InWorldSpace), FromViewToPhysicalSpace(p.vertecies[2].Point\_InWorldSpace));

e.Surface.DrawLine(Color.Green.ToArgb(), FromViewToPhysicalSpace(p.vertecies[2].Point\_InWorldSpace), FromViewToPhysicalSpace(p.vertecies[0].Point\_InWorldSpace));

}

}

if (IsLightSource){

DVector2 LightInPhysicalSpace = FromViewToPhysicalSpace(LightPos\_InWorldSpace);

e.Surface.DrawTriangle(Color.Orange.ToArgb(), new DVector2(LightInPhysicalSpace.X, LightInPhysicalSpace.Y), new DVector2(LightInPhysicalSpace.X + 20, LightInPhysicalSpace.Y), new DVector2(LightInPhysicalSpace.X + 10, LightInPhysicalSpace.Y + 10));

}

}

}

**Вывод**

В результате выполнения данной лабораторной работы я познакомился с возможностью отрисовки фотореалистичных изображений посредством вычисления интенсивностей цвета для вершин или полигонов фигуры в зависимости от применяемой модели освещения - плоской или Гуро.

Лабораторная работа №4-6 по курсу «Компьютерная графика»

Тема: Ознакомление с технологией OpenGL и языком GLSL.  
 Создание шейдерных анимационных эффектов в OpenGL 2.1.

**Постановка задачи**

**Задание**: 1)Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.№3, изобразить заданное тело (то же, что и в л.р. №3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.  
2)Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта:  
Вариант: 5. Анимация. Координата X изменяется по закону X\*cos(t), координата Y изменяется по закону   
Y = Ysin(X+t).

**Решение задачи**

Возьмем за основу способ генерации фигуры и вычисление интенсивностей цвета такие же, какие они былы в Л.Р. 3. При этом важно понимать, что затенение Фонга отличается от Гуро тем, что при его использовании для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей - это будет осуществляться автоматически при загрузке нормалей как выходные атрибуты в вершинном шейдере OpenGL. Стоит также отметить, что все вычисления, связанные с освещением, будут происходить исключительно во фрагментном шейдере OpenGL. Для корректной отрисовки фигуры в среде OpenGL необходимо задать матрицу проекций (свойство OpenGL.GL\_PROJECTION) и объектно-видовую матрицу (свойство OpenGL.GL\_MODELVIEW) - их следует предварительно вычислить и только затем подгрузить в OpenGL. После задания этих матриц будет автоматически производиться следующее преобразование для координат некоторой вершины P :



Чтобы вся информация о фигуре хранилось исключительно в GPU необходимо загружать её в буфер вершин VBO. При этом для вызова метода отрисовки OpenGL необходимо знать, в каком порядке следуют вершины в массиве. При работе с прямым эллиптическим цилиндром будем использоваться порядок обхода GL\_TRIANGLE\_FAN - для оснований, и GL\_TRIANGLE\_STRIP - для боковой поверхности (вызывая этот метод отрисовки таким обходом для каждого этажа соответственно).

Для создания требуемого шейдерного эффекта достаточно немного изменить вершинный шейдер - менять указанным способом координаты Y загружаемых вершин:

Y = Y · cos(t + Y )

Обновленная величина времени t будет подгружаться в шейдер каждый раз при выполнении основного цикла работы программы. Стоит отметить, что при изменении координат вершин не требуется пересчитывать все ранее вычисленные нормали фигуры - изменения при анимации незначительные и в принципе этого не требуют.

**Листинг программы**

**Program.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Runtime.InteropServices;

using System.Text;

using System.Windows.Forms;

using CGLabPlatform;

using SharpGL;

using Device = CGLabPlatform.OGLDevice;

using DeviceArgs = CGLabPlatform.OGLDeviceUpdateArgs;

using CGApplication = MyApp;

public abstract class MyApp : OGLApplicationTemplate<MyApp>{

public enum Visualization{

[Description("Каркасная визуализация")]

NoPolygons,

[Description("Одним цветом")] OneColor,

[Description("Случайные цвета")] RandomColor,

[Description("Метод затенения Фонга")] PhongShading

}

public DVector2 \_ClippingPlanes;

private Commands \_Commands = Commands.FigureChange;

public uint[] indexBuffer = new uint[1];

public uint[] indices;

private List<List<Vertex>> LLvertices;

public uint[] normalDataBuffer = new uint[1];

public uint[] normalIndexBuffer = new uint[1];

public uint[] normalIndices;

public DVector4[] normalPoints;

public DMatrix4 pMatrix;

public uint[] vertexBuffer = new uint[1];

public Vertex[] vertices;

[DisplayEnumListProperty(Visualization.OneColor, "Способ отрисовки")]

public abstract Visualization CurVisual{ get; set; }

[DisplayCheckerProperty(false, "Выполнять анимацию")]

public abstract bool isAnimation{ get; set; }

[DisplayNumericProperty(1000, Minimum: 1, Increment: 1, Name: "Скорость анимации")]

public abstract int TimeSpeed{ get; set; }

#region Работа со светом

public void UpdateLightValues(DeviceArgs e){

var gl = e.gl;

gl.Uniform3(uniform\_Ka\_Material, (float) Ka\_Material.X, (float) Ka\_Material.Y, (float) Ka\_Material.Z);

gl.Uniform3(uniform\_Kd\_Material, (float) Kd\_Material.X, (float) Kd\_Material.Y, (float) Kd\_Material.Z);

gl.Uniform3(uniform\_Ks\_Material, (float) Ks\_Material.X, (float) Ks\_Material.Y, (float) Ks\_Material.Z);

gl.Uniform1(uniform\_P\_Material, (float) P\_Material);

gl.Uniform3(uniform\_Ia\_Material, (float) Ia\_Material.X, (float) Ia\_Material.Y, (float) Ia\_Material.Z);

gl.Uniform3(uniform\_Il\_Material, (float) Il\_Material.X, (float) Il\_Material.Y, (float) Il\_Material.Z);

var LightPosV4 = new DVector4(LightPos, 1);

LightPos\_InWorldSpace = \_PointTransform \* LightPosV4;

gl.Uniform3(uniform\_LightPos, (float) LightPos\_InWorldSpace.X, (float) LightPos\_InWorldSpace.Y,

(float) LightPos\_InWorldSpace.Z);

gl.Uniform2(uniform\_Parameters, (float) Parameters.X, (float) Parameters.Y);

gl.Uniform3(uniform\_CameraPos, (float) CameraPos.X, (float) CameraPos.Y, (float) CameraPos.Z);

gl.Uniform3(uniform\_FragColor, (float) MaterialColor.X, (float) MaterialColor.Y, (float) MaterialColor.Z);

if (isAnimation)

gl.Uniform1(uniform\_time, cur\_time);

else

gl.Uniform1(uniform\_time, -1f);

}

#endregion

protected override void OnMainWindowLoad(object sender, EventArgs args){

ValueStorage.Font = new Font("Arial", 12f);

ValueStorage.ForeColor = Color.Firebrick;

ValueStorage.RowHeight = 30;

ValueStorage.BackColor = Color.BlanchedAlmond;

MainWindow.BackColor = Color.DarkGoldenrod;

ValueStorage.RightColWidth = 50;

VSPanelWidth = 400;

VSPanelLeft = true;

MainWindow.Size = new Size(2500, 1380);

MainWindow.StartPosition = FormStartPosition.Manual;

MainWindow.Location = Point.Empty;

RenderDevice.MouseMoveWithRightBtnDown += (s, e)

=> Offset += new DVector3(0.001 \* Math.Abs(\_Scale.X) \* e.MovDeltaX,

0.001 \* Math.Abs(\_Scale.Y) \* e.MovDeltaY, 0);

RenderDevice.MouseMoveWithLeftBtnDown += (s, e)

=> Rotations += new DVector3(0.1 \* e.MovDeltaY, 0.1 \* e.MovDeltaX, 0);

RenderDevice.MouseWheel += (s, e) => Scale += new DVector3(0.001 \* e.Delta, 0.001 \* e.Delta, 0.001 \* e.Delta);

RenderDevice.Resize += (o, eventArgs) => { \_Commands |= Commands.ChangeProjectionMatrix; };

RenderDevice.VSync = 1;

RenderDevice.AddScheduleTask((gl, s) => {

gl.Enable(OpenGL.GL\_CULL\_FACE);

gl.CullFace(OpenGL.GL\_BACK);

gl.FrontFace(OpenGL.GL\_CW);

gl.ClearColor(0, 0, 0, 0);

gl.Enable(OpenGL.GL\_DEPTH\_TEST);

gl.DepthFunc(OpenGL.GL\_LEQUAL);

gl.ClearDepth(1.0f);

gl.ClearStencil(0);

gl.GenBuffers(1, vertexBuffer);

gl.GenBuffers(1, indexBuffer);

gl.GenBuffers(1, LightVertexBuffer);

gl.GenBuffers(1, LightIndexBuffer);

gl.GenBuffers(1, normalDataBuffer);

gl.GenBuffers(1, normalIndexBuffer); });

#region Загрузка и комплиция шейдера ------------------

RenderDevice.AddScheduleTask((gl, s) => {

var parameters = new int[1];

prog\_shader = gl.CreateProgram();

if (prog\_shader == 0)

throw new Exception("OpenGL Error: не удалось создать шейдерную программу");

var load\_and\_compile = new Func<uint, string, uint>(

(shader\_type, shader\_name) => {

var shader = gl.CreateShader(shader\_type);

if (shader == 0)

throw new Exception("OpenGL Error: не удалось создать объект шейдера");

var source = HelpUtils.GetTextFileFromRes(shader\_name);

gl.ShaderSource(shader, source);

gl.CompileShader(shader);

gl.GetShader(shader, OpenGL.GL\_COMPILE\_STATUS, parameters);

if (parameters[0] != OpenGL.GL\_TRUE){

gl.GetShader(shader, OpenGL.GL\_INFO\_LOG\_LENGTH, parameters);

var stringBuilder = new StringBuilder(parameters[0]);

gl.GetShaderInfoLog(shader, parameters[0], IntPtr.Zero, stringBuilder);

Debug.WriteLine("\n\n\n\n ====== SHADER GL\_COMPILE\_STATUS: ======");

Debug.WriteLine(stringBuilder);

Debug.WriteLine("==================================");

throw new Exception("OpenGL Error: ошибка во при компиляции " + (

shader\_type == OpenGL.GL\_VERTEX\_SHADER ? "вершиного шейдера"

: shader\_type == OpenGL.GL\_FRAGMENT\_SHADER ? "фрагментного шейдера"

: "какого-то еще шейдера"));

}

gl.AttachShader(prog\_shader, shader);

return shader;});

vert\_shader = load\_and\_compile(OpenGL.GL\_VERTEX\_SHADER, "shader1.vert");

frag\_shader = load\_and\_compile(OpenGL.GL\_FRAGMENT\_SHADER, "shader2.frag");

gl.LinkProgram(prog\_shader);

gl.GetProgram(prog\_shader, OpenGL.GL\_LINK\_STATUS, parameters);

if (parameters[0] != OpenGL.GL\_TRUE){

gl.GetProgram(prog\_shader, OpenGL.GL\_INFO\_LOG\_LENGTH, parameters);

var stringBuilder = new StringBuilder(parameters[0]);

gl.GetProgramInfoLog(prog\_shader, parameters[0], IntPtr.Zero, stringBuilder);

Debug.WriteLine("\n\n\n\n ====== PROGRAM GL\_LINK\_STATUS: ======");

Debug.WriteLine(stringBuilder);

Debug.WriteLine("==================================");

throw new Exception("OpenGL Error: ошибка линковкой"); }});

#endregion

#region Удаление шейдера

RenderDevice.Closed += (s, e) => RenderDevice.AddScheduleTask((gl, \_s) => {gl.UseProgram(0); gl.DeleteProgram(prog\_shader); gl.DeleteShader(vert\_shader); gl.DeleteShader(frag\_shader);});

#endregion

#region Связывание аттрибутов и юниформ ------------------

RenderDevice.AddScheduleTask((gl, s) => {

/\* Использующиеся в shader2.frag \*/

uniform\_Ka\_Material = gl.GetUniformLocation(prog\_shader, "Ka\_Material");

if (uniform\_Ka\_Material < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Ka\_Material");

uniform\_Kd\_Material = gl.GetUniformLocation(prog\_shader, "Kd\_Material");

if (uniform\_Kd\_Material < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Kd\_Material");

uniform\_Ks\_Material = gl.GetUniformLocation(prog\_shader, "Ks\_Material");

if (uniform\_Ks\_Material < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Ks\_Material");

uniform\_P\_Material = gl.GetUniformLocation(prog\_shader, "P\_Material");

if (uniform\_P\_Material < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут P\_Material");

uniform\_Ia\_Material = gl.GetUniformLocation(prog\_shader, "Ia\_Material");

if (uniform\_Ia\_Material < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Ia\_Material");

uniform\_Il\_Material = gl.GetUniformLocation(prog\_shader, "Il\_Material");

if (uniform\_Il\_Material < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Il\_Material");

uniform\_LightPos = gl.GetUniformLocation(prog\_shader, "LightPos");

if (uniform\_LightPos < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут LightPos");

uniform\_Parameters = gl.GetUniformLocation(prog\_shader, "Parameters");

if (uniform\_Parameters < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Parameters");

uniform\_CameraPos = gl.GetUniformLocation(prog\_shader, "CameraPos");

if (uniform\_CameraPos < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут CameraPos");

/\* Использующиеся в shader1.vert \*/

attribute\_normale = gl.GetAttribLocation(prog\_shader, "Normal");

if (attribute\_normale < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Normal");

attribute\_coord = gl.GetAttribLocation(prog\_shader, "Coord");

if (attribute\_coord < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Coord");

uniform\_Projection = gl.GetUniformLocation(prog\_shader, "Projection");

if (uniform\_Projection < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Projection");

uniform\_ModelView = gl.GetUniformLocation(prog\_shader, "ModelView");

if (uniform\_ModelView < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут ModelView");

uniform\_FragColor = gl.GetUniformLocation(prog\_shader, "FragColor");

if (uniform\_FragColor < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут FragColor");

uniform\_NormalMatrix = gl.GetUniformLocation(prog\_shader, "NormalMatrix");

if (uniform\_NormalMatrix < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут NormalMatrix");

uniform\_PointMatrix = gl.GetUniformLocation(prog\_shader, "PointMatrix");

if (uniform\_PointMatrix < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут PointMatrix");

uniform\_time = gl.GetUniformLocation(prog\_shader, "Time");

if (uniform\_time < 0)

throw new Exception("OpenGL Error: не удалость связать аттрибут Time");

});

#endregion

}

protected override void OnDeviceUpdate(object s, DeviceArgs e){

var gl = e.gl;

cur\_time += e.Delta \* TimeSpeed / 1e6f;

// Очиста буфера экрана и буфера глубины (иначе рисоваться будет поверх старого )

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT | OpenGL.GL\_STENCIL\_BUFFER\_BIT);

// Задание проекционной и объектно-видовой матрицы

if (0 != ((int) \_Commands & (int) Commands.ChangeProjectionMatrix)){

\_Commands ^= Commands.ChangeProjectionMatrix;

UpdateProjectionMatrix(e);

\_Commands |= Commands.Transform;

}

if (0 != ((int) \_Commands & (int) Commands.NewFigure)){

\_Commands ^= Commands.NewFigure;

Create();

\_Commands |= Commands.FigureChange;

}

if (0 != ((int) \_Commands & (int) Commands.FigureChange)){

\_Commands ^= Commands.FigureChange;

Generate(e);

}

if (0 != ((int) \_Commands & (int) Commands.Transform)){

\_Commands ^= Commands.Transform;

UpdateModelViewMatrix(e);

}

if (0 != ((int) \_Commands & (int) Commands.ChangeLightPos)) // Здесь загружаются данные света

{

\_Commands ^= Commands.ChangeLightPos;

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, LightVertexBuffer[0]);

var LightPosV4 = new DVector4(LightPos, 1);

unsafe{

LightVertexArray = LightPosV4.ToArray();

fixed (double\* ptr = &LightVertexArray[0]){

gl.BufferData(OpenGL.GL\_ARRAY\_BUFFER, LightVertexArray.Length \* sizeof(double), (IntPtr) ptr,

OpenGL.GL\_STATIC\_DRAW);

}

}

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, LightIndexBuffer[0]);

unsafe{

fixed (uint\* ptr = &LightIndexValues[0]){

gl.BufferData(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, LightIndexValues.Length \* sizeof(uint), (IntPtr) ptr,

OpenGL.GL\_STATIC\_DRAW);

}

}

LightPos\_InWorldSpace = \_PointTransform \* LightPosV4;

}

// Задание способа визуализации

if (CurVisual == Visualization.OneColor){

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_FILL);

gl.Color(MaterialColor.X, MaterialColor.Y, MaterialColor.Z);

}

else if (CurVisual == Visualization.RandomColor){

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_FILL);

gl.EnableClientState(OpenGL.GL\_COLOR\_ARRAY);

unsafe{

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, vertexBuffer[0]);

gl.ColorPointer(3, OpenGL.GL\_BYTE, sizeof(Vertex), (IntPtr) (sizeof(float) \* 8));

}

}

else if (CurVisual == Visualization.NoPolygons){

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_LINE);

gl.Color(MaterialColor.X, MaterialColor.Y, MaterialColor.Z);

}

else if (CurVisual == Visualization.PhongShading){

gl.PolygonMode(OpenGL.GL\_FRONT, OpenGL.GL\_FILL);

}

/\* Непосредственная отрисовка \*/

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, vertexBuffer[0]);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, indexBuffer[0]);

if (CurVisual == Visualization.PhongShading){

gl.UseProgram(prog\_shader);

UpdateLightValues(e);

gl.UniformMatrix4(uniform\_ModelView, 1, true, ConvertToFloatArray(ModelViewMatrix));

gl.UniformMatrix4(uniform\_Projection, 1, true, ConvertToFloatArray(pMatrix));

gl.UniformMatrix4(uniform\_NormalMatrix, 1, true, ConvertToFloatArray(NormalMatrix));

gl.UniformMatrix4(uniform\_PointMatrix, 1, true, ConvertToFloatArray(\_PointTransform));

gl.EnableVertexAttribArray((uint) attribute\_normale);

gl.EnableVertexAttribArray((uint) attribute\_coord);

unsafe{

gl.VertexAttribPointer((uint) attribute\_normale, 4, OpenGL.GL\_FLOAT, false, sizeof(Vertex),

(IntPtr) (4 \* sizeof(float)));

gl.VertexAttribPointer((uint) attribute\_coord, 4, OpenGL.GL\_FLOAT, false, sizeof(Vertex), (IntPtr) 0);

}

}else{

gl.UseProgram(0);

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

unsafe{

gl.VertexPointer(4, OpenGL.GL\_FLOAT, sizeof(Vertex), (IntPtr) 0);

}

}

gl.DrawElements(OpenGL.GL\_TRIANGLE\_FAN, (int) after\_top, OpenGL.GL\_UNSIGNED\_INT, (IntPtr) 0);

for (var i = 0; i < (approx1 - 2) / 2 - 1; ++i)

gl.DrawElements(OpenGL.GL\_TRIANGLE\_STRIP, (int) (after\_side - after\_top) / ((approx1 - 2) / 2 - 1),

OpenGL.GL\_UNSIGNED\_INT, (IntPtr) ((

after\_top + (after\_side - after\_top) / ((approx1 - 2) / 2 - 1) \* i) \* sizeof(uint)));

gl.DrawElements(OpenGL.GL\_TRIANGLE\_FAN, (int) (after\_bottom - after\_side), OpenGL.GL\_UNSIGNED\_INT,

(IntPtr) (after\_side \* sizeof(uint)));

if (CurVisual == Visualization.PhongShading){

gl.DisableVertexAttribArray((uint) attribute\_normale);

gl.DisableVertexAttribArray((uint) attribute\_coord);

gl.UseProgram(0);

}else{

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

}

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, 0);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, 0);

if (CurVisual == Visualization.RandomColor) gl.DisableClientState(OpenGL.GL\_COLOR\_ARRAY);

if (isLightActive){

gl.Color(0.99, 0, 0);

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, LightVertexBuffer[0]);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, LightIndexBuffer[0]);

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

unsafe{

gl.VertexPointer(4, OpenGL.GL\_DOUBLE, sizeof(DVector4), (IntPtr) 0);

}

gl.PointSize(10);

gl.DrawElements(OpenGL.GL\_POINTS, 1, OpenGL.GL\_UNSIGNED\_INT, (IntPtr) 0);

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, 0);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, 0);

}

if (isNormalActive){

gl.Color(0, 0, 0.99);

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, normalDataBuffer[0]);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, normalIndexBuffer[0]);

gl.EnableClientState(OpenGL.GL\_VERTEX\_ARRAY);

unsafe{

gl.VertexPointer(4, OpenGL.GL\_DOUBLE, sizeof(DVector4), (IntPtr) 0);

}

gl.DrawElements(OpenGL.GL\_LINES, normalPoints.Length, OpenGL.GL\_UNSIGNED\_INT, (IntPtr) 0);

gl.DisableClientState(OpenGL.GL\_VERTEX\_ARRAY);

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, 0);

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, 0);

}

}

[Flags]

private enum Commands{

None = 0,

Transform = 1 << 0,

FigureChange = 1 << 1,

NewFigure = 1 << 3,

ChangeProjectionMatrix = 1 << 4,

ShadingChange = 1 << 5,

ChangeLightPos = 1 << 6

}

#region Классы

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct Vertex{

public Vertex(float px, float py, float pz, float pw, byte pr, byte pg, byte pb){

vx = px;vy = py;vz = pz;vw = pw;

nx = 0;ny = 0;nz = 0;nw = 0;

r = pr; g = pg; b = pb;

}

public void SetNorm(float nx, float ny, float nz, float nw){

this.nx = nx;

this.nx = ny;

this.nx = nz;

this.nx = nw;

}

public readonly float vx, vy, vz, vw;

public float nx, ny, nz, nw;

public readonly byte r, g, b;

}

private void ChangeNormale(ref Vertex vertex, DVector4 normale){

var check = new DVector4(vertex.nx, vertex.ny, vertex.nz, 0);

check += normale;

vertex.nx = (float) check.X;

vertex.ny = (float) check.Y;

vertex.nz = (float) check.Z;

}

#endregion

#region Работа с афинными преобразованиями фигуры

#region Свойства

private DMatrix4 \_PointTransform = new DMatrix4(new[]{ 1d, 0, 0, 0, 0, 1d, 0, 0,0, 0, 1d, 0,0, 0, 0, 1d});

private DVector3 \_Rotations;

private DVector3 \_Offset;

private DVector3 \_Scale;

[DisplayNumericProperty(1, 0.1, Minimum: 1, Name: "Радиус")]

public float Radius{

get => Get<float>();

set{

if (Set(value)) \_Commands |= Commands.FigureChange;

}

}

[DisplayNumericProperty(new[]{16d, 12d}, Minimum: 4d, Increment: 2d, Name: "Апроксимация")]

public DVector2 Approximation{

get => Get<DVector2>();

set{

if (Set(value)) \_Commands |= Commands.NewFigure;

}

}

public int approx0, approx1;

[DisplayNumericProperty(new[]{0d, 0d, 0d}, 0.1, "Смещение")]

public DVector3 Offset{

get => \_Offset;

set{

if (Set(value)){

\_Offset = value;

UpdateTransformMatrix();

}

}

}

[DisplayNumericProperty(new[]{0d, 0d, 0d}, 1, "Поворот")]

public DVector3 Rotations{

get => \_Rotations;

set{

if (Set(value)){

\_Rotations = value;

if (\_Rotations.X >= 360) \_Rotations.X -= 360;

if (\_Rotations.Y >= 360) \_Rotations.Y -= 360;

if (\_Rotations.Z >= 360) \_Rotations.Z -= 360;

if (\_Rotations.X < 0) \_Rotations.X += 360;

if (\_Rotations.Y < 0) \_Rotations.Y += 360;

if (\_Rotations.Z < 0) \_Rotations.Z += 360;

UpdateTransformMatrix();

}

}

}

[DisplayNumericProperty(new[]{1d, 1d, 1d}, Minimum: 0.1d, Increment: 0.1, Name: "Масштаб")]

public DVector3 Scale{

get => \_Scale;

set{

\_Scale = value;

if (Set(value)) UpdateTransformMatrix();

}

}

public enum Projection{

[Description("Не задана")] NotSet,

[Description("Вид спереди")] InFront,

[Description("Вид сверху")] Above,

[Description("Вид сбоку")] Sideway,

[Description("Изометрия")] Isometry

}

[DisplayEnumListProperty(Projection.NotSet, "Проекция")]

public Projection CurProjection{

get => Get<Projection>();

set{

if (Set(value)){

if (CurProjection == Projection.Above){

\_Scale.Y = 0;

}

else if (CurProjection == Projection.InFront){

\_Scale.Z = 0;

}

else if (CurProjection == Projection.Sideway){

\_Scale.X = 0;

}

else if (CurProjection == Projection.Isometry){

\_Rotations.X = 35;

\_Rotations.Y = 45;

\_Rotations.Z = 0;

}

UpdateTransformMatrix();

\_Commands |= Commands.Transform;

}

}

}

#endregion

#endregion

#region Свойства освещения

[DisplayCheckerProperty(false, "Показывать источник света")]

public abstract bool isLightActive{ get; set; }

[DisplayCheckerProperty(false, "Показывать нормали вершин")]

public abstract bool isNormalActive{ get; set; }

[DisplayNumericProperty(new[]{0.68d, 0.85d, 0.90d}, Minimum: 0d, Maximum: 1d, Increment: 0.01d,

Name: "Цвет материала")]

public DVector3 MaterialColor{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(new[]{0.14d, 0.14d, 0.20d}, Minimum: 0d, Maximum: 1d, Increment: 0.01d,

Name: "Ka материала")]

public DVector3 Ka\_Material{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(new[]{1d, 1d, 0.54d}, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Kd материала")]

public DVector3 Kd\_Material{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(new[]{0.21d, 0.21d, 1d}, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Ks материала")]

public DVector3 Ks\_Material{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(1d, Minimum: 0.01d, Maximum: 100d, Increment: 0.01d, Name: "p материала")]

public double P\_Material{

get => Get<double>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(new[]{1d, 1d, 1d}, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Ia освещения")]

public DVector3 Ia\_Material{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(new[]{1d, 0.5d, 0d}, Minimum: 0d, Maximum: 1d, Increment: 0.01d, Name: "Il освещения")]

public DVector3 Il\_Material{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(new[]{2.5d, 0.5d, 2d}, 0.01d, "Pos освещения")]

public DVector3 LightPos{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ChangeLightPos;

}

}

public DVector4 LightPos\_InWorldSpace;

public uint[] LightVertexBuffer = new uint[1];

public uint[] LightIndexBuffer = new uint[1];

public uint[] LightIndexValues = new uint[1]{0};

public double[] LightVertexArray;

[DisplayNumericProperty(new[]{0.1d, 0.35d}, Minimum: 0.01d, Maximum: 100d, Increment: 0.01d, Name: "md, mk")]

public DVector2 Parameters{

get => Get<DVector2>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

#endregion

#region Свойства отображения

[DisplayNumericProperty(new[]{0d, 0d, 0d}, Minimum: 0d, Increment: 0.01d, Name: "Положение камеры")]

public virtual DVector3 CameraPos{

get => Get<DVector3>();

set{

if (Set(value)) \_Commands |= Commands.ShadingChange;

}

}

[DisplayNumericProperty(-1.7d, Maximum: -0.1d, Increment: 0.1d, Decimals: 2, Name: "Удаленность камеры")]

public virtual double CameraDistance{

get => Get<double>();

set{

if (Set(value)) \_Commands |= Commands.Transform;

}

}

[DisplayNumericProperty(60d, Maximum: 90d, Minimum: 30d, Increment: 1d, Decimals: 1, Name: "Поле зрения")]

public virtual double FieldVision{

get => Get<double>();

set{

if (Set(value)) \_Commands |= Commands.ChangeProjectionMatrix;

}

}

[DisplayNumericProperty(new[]{0.1d, 100d}, Maximum: 1000d, Minimum: 0.1d, Increment: 0.1d, Decimals: 2,

Name: "Плоскости отсечения")]

public DVector2 ClippingPlanes{

get => \_ClippingPlanes;

set{

if (Set(value)){

\_ClippingPlanes = value;

if (ClippingPlanes.X > ClippingPlanes.Y) \_ClippingPlanes.X = \_ClippingPlanes.Y;

\_Commands |= Commands.ChangeProjectionMatrix;

}

}

}

#endregion

#region Поля для работы с шейдерами

private uint prog\_shader;

private uint vert\_shader, frag\_shader;

private int uniform\_Ka\_Material, uniform\_Kd\_Material, uniform\_Ks\_Material;

private int uniform\_P\_Material;

private int uniform\_Ia\_Material, uniform\_Il\_Material;

private int uniform\_LightPos, uniform\_Parameters, uniform\_CameraPos;

private int uniform\_FragColor;

private int uniform\_Projection, uniform\_ModelView, uniform\_NormalMatrix, uniform\_PointMatrix;

private int attribute\_normale, attribute\_coord;

private int uniform\_time;

private float cur\_time;

#endregion

#region Методы

private void UpdateTransformMatrix(){

\_PointTransform = new DMatrix4(new double[]{1, 0, 0, 0,0, 1, 0, 0, 0, 0, 1, 0,0, 0, 0, 1 });

/\* Смещение \*/

var offset = new DMatrix4(new[]{1, 0, 0, -\_Offset.X,0, 1, 0, -\_Offset.Y,0, 0, 1, \_Offset.Z,0, 0, 0, 1});

\_PointTransform \*= offset;

var x\_rotate = new DMatrix4(new[]{1, 0, 0, 0,0, Math.Cos(Math.PI / 180 \* \_Rotations.X), Math.Sin(Math.PI / 180 \* \_Rotations.X), 0,0, -Math.Sin(Math.PI / 180 \* \_Rotations.X), Math.Cos(Math.PI / 180 \* \_Rotations.X), 0,0, 0, 0, 1});

var y\_rotate = new DMatrix4(new[]{Math.Cos(Math.PI / 180 \* \_Rotations.Y), 0, Math.Sin(Math.PI / 180 \* \_Rotations.Y), 0,

0, 1, 0, 0,-Math.Sin(Math.PI / 180 \* \_Rotations.Y), 0, Math.Cos(Math.PI / 180 \* \_Rotations.Y), 0,0, 0, 0, 1});

var z\_rotate = new DMatrix4(new[]{Math.Cos(Math.PI / 180 \* \_Rotations.Z), -Math.Sin(Math.PI / 180 \* \_Rotations.Z), 0, 0,

Math.Sin(Math.PI / 180 \* \_Rotations.Z), Math.Cos(Math.PI / 180 \* \_Rotations.Z), 0, 0,0, 0, 1, 0,0, 0, 0, 1});

\_PointTransform \*= x\_rotate;

\_PointTransform \*= y\_rotate;

\_PointTransform \*= z\_rotate;

/\* Маштабирование \*/

\_PointTransform \*= new DMatrix4(new[]{

\_Scale.X, 0, 0, 0,

0, \_Scale.Y, 0, 0,

0, 0, \_Scale.Z, 0,

0, 0, 0, 1

});

\_Commands |= Commands.Transform;

}

public uint after\_top;

public uint after\_side;

public uint after\_bottom;

private void Create(){

approx0 = (int) Approximation[0]; // меридианы

approx1 = (int) Approximation[1]; //параллели

normalPoints = new DVector4[(approx0 \* (approx1 - 2) / 2 + 2) \* 2];

normalIndices = new uint[(approx0 \* (approx1 - 2) / 2 + 2) \* 2];

}

private void Generate(DeviceArgs e){

LLvertices = new List<List<Vertex>>();

var phi = 2 \* Math.PI / approx0; // горизонтально

var theta = 2 \* Math.PI / approx1; // вертикально

double sumphi = 0;

double sumtheta = 0;

var indx = 0;

var random = new Random();

for (var i = 0; i < approx1 / 2 - 1; i++){

sumtheta += theta;

LLvertices.Add(new List<Vertex>());

for (var j = 0; j < approx0; j++){

sumphi += phi;

LLvertices[LLvertices.Count - 1].Add(new Vertex(

(float) (Radius \* Math.Sin(sumtheta) \* Math.Cos(sumphi)),

(float) (Radius \* Math.Sin(sumtheta) \* Math.Sin(sumphi)), (float) (Radius \* Math.Cos(sumtheta)), 1,

(byte) random.Next(256), (byte) random.Next(256), (byte) random.Next(256)));

}

sumphi = 0;

}

LLvertices.Add(new List<Vertex>());

LLvertices[LLvertices.Count - 1].Add(new Vertex(0, 0, -Radius, 1,

(byte) random.Next(256), (byte) random.Next(256), (byte) random.Next(256))); // i - 1 \_\_\_\_\_\_\_\_

LLvertices.Add(new List<Vertex>());

LLvertices[LLvertices.Count - 1].Add(new Vertex(0, 0, Radius, 1,

(byte) random.Next(256), (byte) random.Next(256), (byte) random.Next(256))); // i +++++

vertices = new Vertex[(LLvertices.Count - 2) \* LLvertices[0].Count + 2];

for (var i = 0; i < vertices.Length; ++i) vertices[i] = new Vertex();

/\* Вычисление нормалей вершин верхнего основания \*/

for (var i = 0; i < LLvertices[0].Count - 2; i++){

var vec1 = new DVector4(

LLvertices[0][i].vx - LLvertices[LLvertices.Count - 1][0].vx,

LLvertices[0][i].vy - LLvertices[LLvertices.Count - 1][0].vy,

LLvertices[0][i].vz - LLvertices[LLvertices.Count - 1][0].vz, 0);

var vec2 = new DVector4(

LLvertices[0][i + 1].vx - LLvertices[LLvertices.Count - 1][0].vx,

LLvertices[0][i + 1].vy - LLvertices[LLvertices.Count - 1][0].vy,

LLvertices[0][i + 1].vz - LLvertices[LLvertices.Count - 1][0].vz, 0);

var res = vec1 \* vec2; // vec2 \* vec1;

res.Normalize();

var TmpVert = LLvertices[0][i];

ChangeNormale(ref TmpVert, res);

LLvertices[0][i] = TmpVert;

TmpVert = LLvertices[LLvertices.Count - 1][0];

ChangeNormale(ref TmpVert, res);

LLvertices[LLvertices.Count - 1][0] = TmpVert;

TmpVert = LLvertices[0][i + 1];

ChangeNormale(ref TmpVert, res);

LLvertices[0][i + 1] = TmpVert;

}

var vec3 = new DVector4(

LLvertices[0][LLvertices[0].Count - 1].vx - LLvertices[LLvertices.Count - 1][0].vx,

LLvertices[0][LLvertices[0].Count - 1].vy - LLvertices[LLvertices.Count - 1][0].vy,

LLvertices[0][LLvertices[0].Count - 1].vz - LLvertices[LLvertices.Count - 1][0].vz, 0);

var vec5 = new DVector4(

LLvertices[0][0].vx - LLvertices[LLvertices.Count - 1][0].vx,

LLvertices[0][0].vy - LLvertices[LLvertices.Count - 1][0].vy,

LLvertices[0][0].vz - LLvertices[LLvertices.Count - 1][0].vz, 0);

var res1 = vec3 \* vec5; // vec5 \* vec3;

res1.Normalize();

var TmpVert1 = LLvertices[0][LLvertices[0].Count - 1];

ChangeNormale(ref TmpVert1, res1);

LLvertices[0][LLvertices[0].Count - 1] = TmpVert1;

TmpVert1 = LLvertices[LLvertices.Count - 1][0];

ChangeNormale(ref TmpVert1, res1);

LLvertices[LLvertices.Count - 1][0] = TmpVert1;

TmpVert1 = LLvertices[0][0];

ChangeNormale(ref TmpVert1, res1);

LLvertices[0][0] = TmpVert1;

/\* Вычисление нормалей вершин нижнего основания \*/

for (var i = 0; i < LLvertices[LLvertices.Count - 3].Count - 2; i++){

var vec1 = new DVector4(

LLvertices[LLvertices.Count - 3][i].vx - LLvertices[LLvertices.Count - 2][0].vx,

LLvertices[LLvertices.Count - 3][i].vy - LLvertices[LLvertices.Count - 2][0].vy,

LLvertices[LLvertices.Count - 3][i].vz - LLvertices[LLvertices.Count - 2][0].vz, 0);

var vec2 = new DVector4(

LLvertices[LLvertices.Count - 3][i + 1].vx - LLvertices[LLvertices.Count - 2][0].vx,

LLvertices[LLvertices.Count - 3][i + 1].vy - LLvertices[LLvertices.Count - 2][0].vy,

LLvertices[LLvertices.Count - 3][i + 1].vz - LLvertices[LLvertices.Count - 2][0].vz, 0);

var res = vec2 \* vec1; // vec1 \* vec2;

res.Normalize();

var TmpVert = LLvertices[LLvertices.Count - 3][i];

ChangeNormale(ref TmpVert, res);

LLvertices[LLvertices.Count - 3][i] = TmpVert;

TmpVert = LLvertices[LLvertices.Count - 2][0];

ChangeNormale(ref TmpVert, res);

LLvertices[LLvertices.Count - 2][0] = TmpVert;

TmpVert = LLvertices[LLvertices.Count - 3][i + 1];

ChangeNormale(ref TmpVert, res);

LLvertices[LLvertices.Count - 3][i + 1] = TmpVert;

}

vec3 = new DVector4(

LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1].vx -

LLvertices[LLvertices.Count - 2][0].vx,

LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1].vy -

LLvertices[LLvertices.Count - 2][0].vy,

LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1].vz -

LLvertices[LLvertices.Count - 2][0].vz, 0);

vec5 = new DVector4(

LLvertices[LLvertices.Count - 3][0].vx - LLvertices[LLvertices.Count - 2][0].vx,

LLvertices[LLvertices.Count - 3][0].vy - LLvertices[LLvertices.Count - 2][0].vy,

LLvertices[LLvertices.Count - 3][0].vz - LLvertices[LLvertices.Count - 2][0].vz, 0);

res1 = vec5 \* vec3; // vec3 \* vec5;

res1.Normalize();

TmpVert1 = LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1];

ChangeNormale(ref TmpVert1, res1);

LLvertices[LLvertices.Count - 3][LLvertices[LLvertices.Count - 3].Count - 1] = TmpVert1;

TmpVert1 = LLvertices[LLvertices.Count - 2][0];

ChangeNormale(ref TmpVert1, res1);

LLvertices[LLvertices.Count - 2][0] = TmpVert1;

TmpVert1 = LLvertices[LLvertices.Count - 3][0];

ChangeNormale(ref TmpVert1, res1);

LLvertices[LLvertices.Count - 3][0] = TmpVert1;

/\* Вычисление нормалей вершин бокового основания \*/

for (var i = 0; i < LLvertices.Count - 3; i++)

for (var j = 0; j < LLvertices[i].Count - 1; j++){

var vec1 = new DVector4(

LLvertices[i + 1][j + 1].vx - LLvertices[i][j + 1].vx,

LLvertices[i + 1][j + 1].vy - LLvertices[i][j + 1].vy,

LLvertices[i + 1][j + 1].vz - LLvertices[i][j + 1].vz, 0);

var vec2 = new DVector4(

LLvertices[i][j].vx - LLvertices[i][j + 1].vx,

LLvertices[i][j].vy - LLvertices[i][j + 1].vy,

LLvertices[i][j].vz - LLvertices[i][j + 1].vz, 0);

var normale1 = vec2 \* vec1;

normale1.Normalize();

var TmpVert = LLvertices[i][j + 1];

ChangeNormale(ref TmpVert, normale1);

LLvertices[i][j + 1] = TmpVert;

TmpVert = LLvertices[i][j];

ChangeNormale(ref TmpVert, normale1);

LLvertices[i][j] = TmpVert;

TmpVert = LLvertices[i + 1][j + 1];

ChangeNormale(ref TmpVert, normale1);

LLvertices[i + 1][j + 1] = TmpVert;

vec1 = -vec1;

vec2 = -vec2;

var normale2 = vec2 \* vec1;

normale2.Normalize();

TmpVert = LLvertices[i][j];

ChangeNormale(ref TmpVert, normale1);

LLvertices[i][j] = TmpVert;

TmpVert = LLvertices[i + 1][j + 1];

ChangeNormale(ref TmpVert, normale1);

LLvertices[i + 1][j + 1] = TmpVert;

TmpVert = LLvertices[i + 1][j];

ChangeNormale(ref TmpVert, normale1);

LLvertices[i + 1][j] = TmpVert;

}

/\* Нормализация нормалей вершин \*/

foreach (var iter in LLvertices)

foreach (var iter1 in iter){

var vec4 = new DVector4(iter1.nx, iter1.ny, iter1.nz, iter1.nw);

vec4.Normalize();

iter1.SetNorm((float) vec4.X, (float) vec4.Y, (float) vec4.Z, (float) vec4.W);

}

var Lindices = new List<uint>();

/\* Полигоны верхнего основания \*/

Lindices.Add((uint) ((LLvertices.Count - 2) \* LLvertices[0].Count + 2) - 1);

for (var i = LLvertices[0].Count - 1; i >= 0; --i) Lindices.Add((uint) i);

Lindices.Add((uint) LLvertices[0].Count - 1);

Lindices.Add((uint) ((LLvertices.Count - 2) \* LLvertices[0].Count + 2) - 1);

after\_top = (uint) Lindices.Count;

/\* Боковые полигоны \*/

indx = 0;

for (var k = 0; k < LLvertices.Count - 3; ++k){

for (var i = 0; i < LLvertices[k].Count - 1; ++i){

Lindices.Add((uint) (indx + LLvertices[0].Count));

Lindices.Add((uint) (indx + 1));

++indx;

}

Lindices.Add((uint) (indx + LLvertices[0].Count));

Lindices.Add((uint) (indx + 1 - LLvertices[0].Count));

Lindices.Add((uint) (indx + 1));

Lindices.Add((uint) (indx + 2 - LLvertices[0].Count));

++indx;

}

after\_side = (uint) Lindices.Count;

/\* Полигоны нижнего основания \*/

Lindices.Add((uint) ((LLvertices.Count - 2) \* LLvertices[0].Count + 2) - 2);

indx = (LLvertices.Count - 3) \* LLvertices[0].Count;

for (var i = 0; i < LLvertices[LLvertices.Count - 3].Count; ++i){

Lindices.Add((uint) indx);

++indx;

}

Lindices.Add((uint) ((LLvertices.Count - 3) \* LLvertices[0].Count));

Lindices.Add((uint) ((LLvertices.Count - 2) \* LLvertices[0].Count + 2) - 2);

after\_bottom = (uint) Lindices.Count;

indices = new uint[Lindices.Count];

var i1 = 0;

foreach (var iter in Lindices){

indices[i1] = iter;

++i1;

}

var art\_it = 0;

foreach (var iter in LLvertices)

foreach (var iter1 in iter){

vertices[art\_it] = iter1;

++art\_it;

}

/\* Инициализация массивов для отрисовки нормалей \*/

var normalLength = 0.25;

for (var i = 0; i < vertices.Length; ++i){

normalPoints[2 \* i] = new DVector4(vertices[i].vx, vertices[i].vy, vertices[i].vz, vertices[i].vw);

normalPoints[2 \* i + 1] = new DVector4(vertices[i].vx + normalLength \* vertices[i].nx,

vertices[i].vy + normalLength \* vertices[i].ny,

vertices[i].vz + normalLength \* vertices[i].nz,

vertices[i].vw + normalLength \* vertices[i].nw);

}

for (var i = 0; i < normalIndices.Length; ++i) normalIndices[i] = (uint) i;

/\* Загрузка буферов \*/

var gl = e.gl;

unsafe{

/\* Обработка массива вершин \*/

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, vertexBuffer[0]);

fixed (Vertex\* ptr = &vertices[0]){

gl.BufferData(OpenGL.GL\_ARRAY\_BUFFER, vertices.Length \* sizeof(Vertex), (IntPtr) ptr,

OpenGL.GL\_STATIC\_DRAW);

}

/\* Обработка индексного массива \*/

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, indexBuffer[0]);

fixed (uint\* ptr = &indices[0]){gl.BufferData(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, indices.Length \* sizeof(uint), (IntPtr) ptr, OpenGL.GL\_STATIC\_DRAW);

}

/\* Обработка массива нормалей \*/

gl.BindBuffer(OpenGL.GL\_ARRAY\_BUFFER, normalDataBuffer[0]);

fixed (DVector4\* ptr = &normalPoints[0]){

gl.BufferData(OpenGL.GL\_ARRAY\_BUFFER, normalPoints.Length \* sizeof(DVector4), (IntPtr) ptr,

OpenGL.GL\_STATIC\_DRAW);

}

/\* Обработка индексного массива нормалей \*/

gl.BindBuffer(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, normalIndexBuffer[0]);

fixed (uint\* ptr = &normalIndices[0]){ gl.BufferData(OpenGL.GL\_ELEMENT\_ARRAY\_BUFFER, normalIndices.Length \* sizeof(uint), (IntPtr) ptr,OpenGL.GL\_STATIC\_DRAW);

}

}

}

private void UpdateProjectionMatrix(DeviceArgs e) {// Задание матрицы проекций

var gl = e.gl;

gl.MatrixMode(OpenGL.GL\_PROJECTION);

pMatrix = Perspective(FieldVision, 1, ClippingPlanes.X, ClippingPlanes.Y);

gl.LoadMatrix(pMatrix.ToArray(true));

}

private DMatrix4 Rotation(double x\_rad, double y\_rad, double z\_rad){

var x\_rotate = new DMatrix4(new[]{ 1, 0, 0, 0, 0, Math.Cos(x\_rad), -Math.Sin(x\_rad), 0,0, Math.Sin(x\_rad), Math.Cos(x\_rad), 0,0, 0, 0, 1});

var y\_rotate=new DMatrix4(new[]{Math.Cos(y\_rad),0, Math.Sin(y\_rad),0,0,1,0,0,-Math.Sin(y\_rad),0,Math.Cos(y\_rad),0,0,0,0,1});

var z\_rotate=new DMatrix4(new[]{Math.Cos(z\_rad),-Math.Sin(z\_rad),0,0,Math.Sin(z\_rad),Math.Cos(z\_rad),0,0,0,0,1,0,0,0,0,1});

return x\_rotate \* y\_rotate \* z\_rotate;

}

public DMatrix4 ModelViewMatrix;

public DMatrix4 NormalMatrix;

private float[] ConvertToFloatArray(DMatrix4 Matrix){

var MatrixF = new float[16];

var MatrixD = Matrix.ToArray();

for (var i = 0; i < 16; ++i) MatrixF[i] = (float) MatrixD[i];

return MatrixF;

}

private void UpdateModelViewMatrix(DeviceArgs e) {

var gl = e.gl;

gl.MatrixMode(OpenGL.GL\_MODELVIEW);

var deg2rad = Math.PI / 180; // Вращается камера, а не сам объект

var cameraTransform = (DMatrix3) Rotation(deg2rad \* CameraPos.X, deg2rad \* CameraPos.Y, deg2rad \* CameraPos.Z);

var cameraPosition = cameraTransform \* new DVector3(0, 0, CameraDistance);

var cameraUpDirection = cameraTransform \* new DVector3(0, 1, 0);

// Мировая матрица (преобразование локальной системы координат в мировую)

var mMatrix = \_PointTransform;

// Видовая матрица (переход из мировой системы координат к системе координат камеры)

var vMatrix = LookAt(DMatrix4.Identity, cameraPosition, DVector3.Zero, cameraUpDirection);

// матрица ModelView

ModelViewMatrix = vMatrix \* mMatrix;

gl.LoadMatrix(ModelViewMatrix.ToArray(true));

// Матрица преобразования вектора

NormalMatrix = DMatrix3.NormalVecTransf(mMatrix);

}

// Построение матрицы перспективной проекции

private static DMatrix4 Perspective(double verticalAngle, double aspectRatio, double nearPlane, double farPlane){

var radians = verticalAngle / 2 \* Math.PI / 180;

var sine = Math.Sin(radians);

if (nearPlane == farPlane || aspectRatio == 0 || sine == 0)

return DMatrix4.Zero;

var cotan = Math.Cos(radians) / sine;

var clip = farPlane - nearPlane;

return new DMatrix4(

cotan / aspectRatio, 0, 0, 0, 0, cotan, 0, 0, 0, 0, -(nearPlane + farPlane) / clip, -(2.0 \* nearPlane \* farPlane) / clip, 0, 0, -1.0, 1.0);

}

// Метод умножения матрицы на видовую матрицу, полученную из точки наблюдения

private static DMatrix4 LookAt(DMatrix4 matrix, DVector3 eye, DVector3 center, DVector3 up){

var forward = (center - eye).Normalized();

if (forward.ApproxEqual(DVector3.Zero, 0.00001))

return matrix;

var side = (forward \* up).Normalized();

var upVector = side \* forward;

var result = matrix \* new DMatrix4(

+side.X, +side.Y, +side.Z, 0,

+upVector.X, +upVector.Y, +upVector.Z, 0,

-forward.X, -forward.Y, -forward.Z, 0,

0, 0, 0, 1

);

result.M14 -= result.M11 \* eye.X + result.M12 \* eye.Y + result.M13 \* eye.Z;

result.M24 -= result.M21 \* eye.X + result.M22 \* eye.Y + result.M23 \* eye.Z;

result.M34 -= result.M31 \* eye.X + result.M32 \* eye.Y + result.M33 \* eye.Z;

result.M44 -= result.M41 \* eye.X + result.M42 \* eye.Y + result.M43 \* eye.Z;

return result;

}

#endregion

}

public abstract class AppMain : CGApplication{

[STAThread]

private static void Main(){

RunApplication();

}

}

**shader1.vert**

#version 150 core

attribute vec4 Normal;

attribute vec4 Coord;

out vec3 FragNormale;

out vec3 FragVertex;

uniform mat4 Projection;

uniform mat4 ModelView;

uniform mat4 NormalMatrix;

uniform mat4 ModelMatrix;

uniform mat4 PointMatrix;

uniform float Time;

void main(void){

vec4 Coord\_now = Coord;

if (Time > 0){

Coord\_now.x = Coord.x \* cos(Time);

Coord\_now.y = Coord.y \* sin(Time + Coord\_now.x);

}

FragVertex = vec3(PointMatrix \* Coord\_now);

FragNormale = vec3(NormalMatrix \* Normal);

FragNormale = normalize(FragNormale);

gl\_Position = (Projection \* ModelView) \* Coord\_now;

}

**shader2.frag**

#version 150 core

in vec3 FragNormale;

in vec3 FragVertex;

uniform vec3 Ka\_Material;

uniform vec3 Kd\_Material;

uniform vec3 Ks\_Material;

uniform float P\_Material;

uniform vec3 Ia\_Material;

uniform vec3 Il\_Material;

uniform vec3 LightPos;

uniform vec2 Parameters;

uniform vec3 CameraPos;

uniform vec3 FragColor;

void main(void){

vec3 L = vec3(LightPos.x - FragVertex.x, LightPos.y - FragVertex.y, LightPos.z - FragVertex.z);

float dist = length(L);

L = normalize(L);

vec3 FragNormaleW = normalize(FragNormale);

float I\_red = Ia\_Material.x \* Ka\_Material.x;

float I\_green = Ia\_Material.y \* Ka\_Material.y;

float I\_blue = Ia\_Material.z \* Ka\_Material.z;

I\_red += clamp(0, 1, Il\_Material.x \* Kd\_Material.x \* dot(L, FragNormaleW) / (Parameters[0] \* dist + Parameters[1]));

I\_green += clamp(0, 1, Il\_Material.y \* Kd\_Material.y \* dot(L, FragNormaleW) / (Parameters[0] \* dist + Parameters[1]));

I\_blue += clamp(0, 1, Il\_Material.z \* Kd\_Material.z \* dot(L, FragNormaleW) / (Parameters[0] \* dist + Parameters[1]));

if (dot(L, FragNormaleW) > 0){

vec3 S = vec3(CameraPos.x - FragVertex.x, CameraPos.y - FragVertex.y, CameraPos.z - FragVertex.z);

vec3 R = vec3(reflect(-L, FragNormale));

S = normalize(S);

R = normalize(R);

if (dot(R, S) > 0){

I\_red += clamp(0, 1, Il\_Material.x \* Ks\_Material.x \* pow(dot(R, S), P\_Material) / (Parameters[0] \* dist + Parameters[1]));

I\_green += clamp(0, 1, Il\_Material.y \* Ks\_Material.y \* pow(dot(R, S), P\_Material) / (Parameters[0] \* dist + Parameters[1]));

I\_blue += clamp(0, 1, Il\_Material.z \* Ks\_Material.z \* pow(dot(R, S), P\_Material) / (Parameters[0] \* dist + Parameters[1]));

}

}

I\_red = min(1, I\_red);

I\_green = min(1, I\_green);

I\_blue = min(1, I\_blue);

vec4 result = vec4(FragColor.x \* I\_red, FragColor.y \* I\_green, FragColor.z \* I\_blue, 1);

gl\_FragColor = result;

}

**Вывод**

В результате выполнения данной лабораторной работы я познакомился с возможностями OpenGL поотрисовки 3D фигур, в частности, использованием буфера вершин, шейдеров, подгружаемых глобальных преобразующих матриц и многого другого. Полученные знания я применил для отрисовки прямого эллиптического цилиндра с затенением по Фонгу.

Лабораторная работа №7 по курсу «Компьютерная графика»

Тема: Построение плоских полиномиальных кривых.

**Постановка задачи**

**Задание**: Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения.  
Вариант: 5. Кривая Безье 2-й степени.

**Решение задачи**

Кривая Безье 2-й степени строится по 3 точкам и имеет вид: B(t) = (1 − t) 2P0 + 2t(1 − t)P1 + t 2P2, где P0, P1, P2 - точки, а t - параметр от 0 до 1. Для построения составной прямой нужно чередовать точки, то есть брать первую вторую третью, третью чевертую пятую и т.д. Также есть возможность двигать ближайшие точки, для это надо пройтись по всем точкам и посчитать расстояние от них до позиции мыши, выбрав ту, у которой это расстояние будет наименьшим.

**Листинг программы**

**Program.cs**

#define UseOpenGL // Раскомментировать для использования OpenGL

#if (!UseOpenGL)

using Device = CGLabPlatform.GDIDevice;

using DeviceArgs = CGLabPlatform.GDIDeviceUpdateArgs;

#else

using Device = CGLabPlatform.OGLDevice;

using DeviceArgs = CGLabPlatform.OGLDeviceUpdateArgs;

using SharpGL;

#endif

using System;

using CGLabPlatform;

using System.Linq;

using System.Drawing;

using System.Windows.Forms;

using CGApplication = MyApp;

using Lab7;

public abstract class MyApp : OGLApplicationTemplate<CGApplication>{

#region Свойства

[DisplayNumericProperty(0.01, Minimum: 0.005, Maximum: 1.0, Increment: 0.005, Name: "dt")]

public abstract float dt{ get; set; }

[DisplayNumericProperty(new[]{-0.5d, -0.5d}, 0.05, "P0")]

public abstract DVector2 P0{ get; set; }

[DisplayNumericProperty(new[]{-0.5d, 0.5d}, 0.05, "P1")]

public abstract DVector2 P1{ get; set; }

[DisplayNumericProperty(new[]{0.5d, 0.5d}, 0.05, "P2")]

public abstract DVector2 P2{ get; set; }

public abstract DVector2 DefaultDIBSize{ get; set; }

private Bezier2Curve curve;

private byte? selectedPoint;

private const float dotRadius = 10f;

#endregion

protected override unsafe void OnMainWindowLoad(object sender, EventArgs args){

ValueStorage.Font = new Font("Arial", 12f);

ValueStorage.ForeColor = Color.Firebrick;

ValueStorage.RowHeight = 30;

ValueStorage.BackColor = Color.BlanchedAlmond;

MainWindow.BackColor = Color.DarkGoldenrod;

ValueStorage.RightColWidth = 50;

VSPanelWidth = 400;

VSPanelLeft = true;

MainWindow.Size = new Size(2500, 1380);

MainWindow.StartPosition = FormStartPosition.Manual;

MainWindow.Location = Point.Empty;

VSPanelWidth = 268;

ValueStorage.RightColWidth = 60;

RenderDevice.VSync = 1;

#region Обработчики событий мыши и клавиатуры

RenderDevice.MouseMoveWithLeftBtnDown += (s, e) => {

if (!selectedPoint.HasValue) return;

var w = RenderDevice.Width / 2.0;

var h = RenderDevice.Height / 2.0;

var ws = WindowScale();

var wsX = RenderDevice.Width / DefaultDIBSize.X;

var wsY = RenderDevice.Height / DefaultDIBSize.Y;

switch (selectedPoint.Value){

case 0:

P0 += new DVector2(e.MovDeltaX \* wsX / w, -e.MovDeltaY \* wsY / h) / ws;

return;

case 1:

P1 += new DVector2(e.MovDeltaX \* wsX / w, -e.MovDeltaY \* wsY / h) / ws;

return;

case 2:

P2 += new DVector2(e.MovDeltaX \* wsX / w, -e.MovDeltaY \* wsY / h) / ws;

return;

}

};

RenderDevice.MouseDown += (s, e) => {

var w = RenderDevice.Width / 2.0;

var h = RenderDevice.Height / 2.0;

var ws = WindowScale();

var wsX = RenderDevice.Width / DefaultDIBSize.X;

var wsY = RenderDevice.Height / DefaultDIBSize.Y;

var hit = new DVector2(

(e.X - w) \* wsX / w,

-(e.Y - h) \* wsY / h

) / ws;

var area = new DVector2(

dotRadius \* wsX / w,

dotRadius \* wsY / h

) / ws;

if (IsHit(P0, hit, area)){

selectedPoint = 0;

return;

}

if (IsHit(P1, hit, area)){

selectedPoint = 1;

return;

}

if (IsHit(P2, hit, area)){

selectedPoint = 2;

return;

}

};

RenderDevice.MouseUp += (s, e) => { selectedPoint = null; };

#endregion

#region Инициализация OGL и параметров рендера

RenderDevice.AddScheduleTask((gl, s) => {

// Фон

gl.ClearColor(0, 0, 0, 0);

// Сглаживание линий

gl.Enable(OpenGL.GL\_LINE\_SMOOTH);

gl.BlendFunc(OpenGL.GL\_LINE\_SMOOTH\_HINT, OpenGL.GL\_NICEST);

});

#endregion

PropertyChanged += (s, e) => {

if (e.PropertyName == nameof(P0)

|| e.PropertyName == nameof(P1)

|| e.PropertyName == nameof(P2)

|| e.PropertyName == nameof(dt)

)

curve = new Bezier2Curve(P0, P1, P2, dt);

};

curve = new Bezier2Curve(P0, P1, P2, dt);

DefaultDIBSize = new DVector2(RenderDevice.Width, RenderDevice.Height);

}

protected override unsafe void OnDeviceUpdate(object s, DeviceArgs e){

if (curve == null) return;

var gl = e.gl;

gl.Clear(

OpenGL.GL\_COLOR\_BUFFER\_BIT |

OpenGL.GL\_DEPTH\_BUFFER\_BIT |

OpenGL.GL\_STENCIL\_BUFFER\_BIT

);

gl.LoadIdentity();

gl.MatrixMode(OpenGL.GL\_PROJECTION);

// Получаем матрицу преобразований

var t = GetTranslateMat();

curve.ApplyTransform(t);

// Касательные

gl.LineWidth(2f);

gl.Begin(OpenGL.GL\_LINE\_STRIP);

gl.Color(0.0f, 0.5f, 0.0f);

foreach (var d in curve.Dots.Select(x => x.pointInWorld)) gl.Vertex(d.X, d.Y);

gl.End();

// Крвая Безье

gl.LineWidth(5f);

gl.Begin(OpenGL.GL\_LINE\_STRIP);

gl.Color(1.0f, 1.0f, 1.0f);

foreach (var p in curve.Points.Select(x => x.pointInWorld)) gl.Vertex(p.X, p.Y);

gl.End();

// Точки P0, P1, P2

gl.PointSize(dotRadius);

gl.Begin(OpenGL.GL\_POINTS);

gl.Color(1.0f, 0.0f, 0.0f);

foreach (var d in curve.Dots.Select(x => x.pointInWorld)) gl.Vertex(d.X, d.Y);

gl.End();

}

private DMatrix3 GetTranslateMat(){

// Формируем матрицу преобразований Translate

var ws = WindowScale();

var wsX = RenderDevice.Width / DefaultDIBSize.X;

var wsY = RenderDevice.Height / DefaultDIBSize.Y;

// Масштабируем при изменении размера окна

var scaleMat = new DMatrix3(ws / wsX, 0, 0, 0, ws / wsY, 0, 0, 0, 1

);

return scaleMat;

}

private double WindowScale(){

return Math.Min(

RenderDevice.Width / DefaultDIBSize.X,

RenderDevice.Height / DefaultDIBSize.Y

);

}

private bool IsHit(DVector2 target, DVector2 hit, DVector2 area){

return

target.X - area.X <= hit.X && hit.X <= target.X + area.X &&

target.Y - area.Y <= hit.Y && hit.Y <= target.Y + area.Y;

}

public abstract class AppMain : CGApplication{

[STAThread]

private static void Main(){

RunApplication();

}

}

}

**Bezier2Curve.cs**

using System.Collections.Generic;

using CGLabPlatform;

namespace Lab7{

public class Bezier2Curve{

public Vertex P0, P1, P2;

public Vertex[] Points;

public Bezier2Curve(DVector2 p0, DVector2 p1, DVector2 p2, double dt){

P0 = new Vertex(p0);

P1 = new Vertex(p1);

P2 = new Vertex(p2);

Points = new Vertex[(int) (1 / dt) + 2];

var i = 0;

for (var t = 0.0; t <= 1; t += dt, i++) Points[i] = new Vertex(Bezier2(p0, p1, p2, t));

Points[i] = new Vertex(Bezier2(p0, p1, p2, 1.0));

}

public void ApplyTransform(DMatrix3 t){

// Вершины

foreach (var p in Points) p.pointInWorld = t \* p.pointInLocalSpace;

P0.pointInWorld = t \* P0.pointInLocalSpace;

P1.pointInWorld = t \* P1.pointInLocalSpace;

P2.pointInWorld = t \* P2.pointInLocalSpace;

}

private DVector2 Bezier2(DVector2 p0, DVector2 p1, DVector2 p2, double t){

return (1.0 - t) \* (1.0 - t) \* p0 + 2 \* t \* (1.0 - t) \* p1 + t \* t \* p2;

}

public IEnumerable<Vertex> Dots{

get{ return new Vertex[]{P0, P1, P2}; }

}

}

public class Vertex{

public readonly DVector3 pointInLocalSpace;

public DVector3 pointInWorld;

public Vertex(DVector2 v){

pointInLocalSpace = new DVector3(v, 1.0);

}

}

}

**Вывод**

В результате выполнения данной лабораторной работы я познакомился построением плоских полиномиальных кривых с использованием технологий OpenGL.